TEX, macro, espansioni, box, LATEX, ambienti, ...

Da Paul W. Abrahams, Stephan Von Bechtolsheim, Victor Eijkhout
Donald E. Knuth
Claudio Beccari, Gianni Gilardi, Enrico Gregorio
Johannes Braams, Leslie Lamport, Frank Mittelbach

Compilatore Riccardo Nisi

27 agosto 2011

Indice

1 Introduzione

Sono da molti anni un utente TeX LaTeX con la curiosità di farsi un'idea di come stanno le cose dietro ai comandi disponibili, che è poi il senso di questa ricerca, ammesso che ce l'abbia.

TeX è in grado di comporre una lettera, una scheda tecnica, un articolo scientifico e molto altro. La contropartita è che anche il più modesto utente deve conoscerlo, almeno fino a quanto lo richiedono le sue esigenze. Le risposte si trovano nelle pagine de *The TEXbook*, spesso in maniera non esplicita e metterle insieme non è facile, tanto da aver fatto abbondare pubblicazioni in grado di facilitare molta parte di quel lavoro. E poi c'è il LaTeX con l'offerta di un quadro applicativo estremamente vasto, arricchito da decine di *classi* e migliaia di *pacchetti di estensioni*. Impossibile conoscere tutto il materiale a disposizione e controllare ciò che copre per cui può diventare irrinunciabile ricorrere a macro o comandi personali con TeX o LaTeX, magari scritti male. Anche se più indirizzato all'utente di TeX o forse proprio per questo, non è semplice cercare di capire come stanno le cose di LaTeX, occorre infatti riferirsi alla sua complessa produzione specialistica. Vediamo come iniziare dal TeX.

Muoversi con un minimo di cognizione in LaTeX significa innanzi tutto conoscere almeno le proprietà essenziali e la struttura delle macro TeX. Ho scelto di farlo riferendomi a macro esistenti cercando di capirle e commentarle. Un approccio non diretto e in un certo qual modo, fatte le traslazioni del caso, simile a quello indiretto ma assai funzionale delle cellule del nostro corpo (vedi ad es.

il messaggio proteico che raggiunta la membrana non la può superare se non modificato in maniera da poter raggiungere la sua destinazione nella cellula).

La sez.2 guarda alle macro senza e con parametri (2.1 - 2.4), definite con `\edef` (2.5), con l'uso di `\noexpand` o `\expandafter` (2.6) e con `\futerelet` (2.8). Nella sezione 3 si guarda a come TEX gestisce il testo di un box e alla sua incorniciatura e a come il testo può distribuirsi nel box con l'ausilio dei registri. La sezione 4, che inizia creando con TEX un ambiente contenente un testo o in box, guarda alla gestione di un ambiente da parte di LATEX: una esplorazione che risente del fatto che l'ho compresa lavorandoci su. Gli esempi di 4.1 si dedicano all'analisi del codice delle dichiarazioni di alcuni `\newenvironment` e quelli di 4.2.1 di alcuni `\newcommand`. L'ultimo argomento TEX è quello dei token della sez.5. Il percorso si conclude (sez.6)con un esempio di interattività in LATEX e la gestione di alcune operazioni sulle figure eseguite tramite il Terminale.

Gli esempi della sezione 2 sono suggeriti da *TEX per l'impaziente* di P.W. Abrahams, K.Berry, K.A.Hargreaves, *TEX in practice* di Stephan Von Bechtolsheim, *TEX by Topic* di Victor Eijkhout, *Il TEXbook* di Donald E. Knuth, *LATEX* di Claudio Beccari, *Il TEX* di Gianni Gilardi e *Appunti di programmazione in LATEX e TEX* di Enrico Gregorio. Per le altre sezioni, in gran parte su LATEX, oltre a quelli indicati localmente i riferimenti sono stati *TEX by Topic*, *TEX per l'impaziente*, *The TEXbook*, *LATEX User's Guide and Reference Manual* di Leslie Lamport, *Appunti di programmazione in LATEX e TEX*, *The LATEX 2ε Sources/2009/477pp.* di J. Braams - D.Carlisle - L. Lamport - F. Mittelbach - C. Rowley - R. Schöpf, *The LATEX Companion* di Frank Mittelbach and Michel Goossens, *Guide to LATEX* di Helmut Kopka and Patrick W.Daly.

Ringrazio poi il prof. Claudio Beccari che con la sua competenza, disponibilità e cortesia, in un lungo percorso e con un gran numero di e-mail, mi ha moltissimo aiutato a gestire e capire i `\newenvironment` (e non solo) e sollecitato verso il TEX. Suoi commenti e note sono presenti nelle sezioni 4 e 5.

2 Macro (da *TEX by Topic*)

Dal punto di vista di TEX una macro è una lista di token semplici o complessi abbreviata in un sequenza di controllo . La definizione della macro in genere inizia con `\def`. In una macro del tipo `\def\abc{\de f \t}` la sequenza di controllo è `\abc` e `\de f \t` è il suo testo. Per capire come lavora una macro occorre tener presente i quattro processi con cui TEX, in una sorta di top-down, passa dal testo immesso all'output finale. I processori generalmente intervengono in sequenza, ma essendo tutti contemporaneamente attivi possono agire in qualsiasi momento, come accade nelle macro con parametri che devono assorbire i relativi argomenti prima di essere espanse. Infatti i parametri delle macro sono contenuti in appositi `stack` così come lo sono tutti i testi di sostituzione delle macro[1]. Con questo livello di conoscenza si possono seguire le espansioni più semplici [2], per i casi più articolati spesso non si riesce ad entrare nei dettagli.

[1] Vedi pag. 300-301 de *TheTEXbook* e 277 di *TEX by Topic*.
[2] Esempi 2.3.1, 2.4.1, 2.4.2, 2.4.3, 2.5.1, 2.6.1, 2.8.1, 2.8.2

1. Il processore di input trasforma i caratteri immessi e le sequenze di controllo in una lista di token . I comandi rappresentati dalle sequenze di controllo saranno processati dai livelli successivi.

2. Il processore di espansione inizia dal primo token e fa la scansione di tutta lista . Se un token non è espandibile passa al successivo altrimenti lo sostituisce con il testo di rimpiazzamento. Macro, condizionali e un certo tipo di primitive saranno espanse. C'è da dire che una macro definita da `\def` va in espansione quando è invocata mentre se è definita con `\edef` l'espansione è contemporanea al processo di input in lista rendendo la macro sensibile al valore che hanno in quella fase le sequenze di controllo incontrate. Nel caso di una macro l'espansione consiste nel sostituire la sequenza di controllo con il relativo testo di definizione detto anche di sostituzione; nel caso visto sopra è `\de f \t`. Se una macro ha parametri i token dell'argomento sono assorbiti dal flusso del processore di esecuzione prima dell'espansione. Nel testo di una struttura condizionale l'espansione elimina tutte le parti non direttamente rilevanti.

3. Il processo di esecuzione . Le sequenze di controllo che non sono espandibili, vedi le primitive, sono eseguite. Assegnazioni, creazione di modi orizzontali, verticali, matematici sono caratteristici di questo livello.

4. Il quarto livello ha competenza sull'output che è un file `.dvi` oggi `.pdf`.

2.1 Macro senza parametri

2.1.1 Da *TeX by Topic*

Nel caso di `\def\abc{AAA BBB CCC}` il processore di input trasforma la macro in una lista di token distinguendo sequenza di controllo e testo. Se il documento invoca la macro si ha la sua espansione , un'operazione in cui il processo di espansione rimpiazza il chiamante `\abc` con una copia del testo di sostituzione
`\abc` $\Rightarrow$ AAA BBB CCC

2.2 Macro e parametri

In una macro , tra la sequenza di controllo e la graffa di apertura del testo di sostituzione possono trovare posto da 1 a 9 parametri , indicati dal carattere `#` e seguiti in maniera strettamente ordinata da un numero da 1 a 9. Tali parametri dovranno essere ripetuti tra le graffe del testo di sostituzione per indicare il corrispondente argomento. I parametri portano nella macro dati messi in campo dal documento in cui si opera. I dati messi in campo con una specifica assegnazione sono gli argomenti che andranno a sostituire i parametri corrispondenti che sono tra le graffe.

2.2.1 Riferendosi a *TEX by Topic*

Date le macro,
▷ `\def\u{*bst 21*}`
▷ `\def\np{32 a 57}`
▷ `\def\xyz#1#2#3{-#1-#2-#3-}`
▷ `\def\foo#1{\count258=4#1\relax}`
si considerano alcune invocazioni:

- `\xyz m nr llz w` qui il testo del chiamante è formato da caratteri e spazi, e la corrispondenza è tra i primi tre caratteri (un carattere è un token di categoria 11, una sequenza di controllo è un token complesso , lo spazio ha categoria 10 e non è considerato un argomento) e i tre parametri, gli eventuali spazi tra questi caratteri sono trascurati. Dopo aver ricevuto gli argomenti la macro viene espansa andando a rimpiazzare con il suo testo aggiornato la sequenza di controllo del chiamante. Occorre considerare che i caratteri non inviati come argomento sono rimasti al loro posto e ora si trovano in coda al testo di sostituzione. La chiamata `\xyz m nr llz w` ⇒ -m-n-r- llz w

- Nella invocazione `\xyz m \np llz \u w` oltre ai caratteri sono presenti chiamate a macro definite in precedenza. I parametri della macro `\xyz` riceveranno gli argomenti `m`, `\np` (è un token complesso e sarà rimpiazzato dal proprio testo di sostituzione), e `l`. A questo punto il testo aggiornato della macro `\xyz` va a sostituire la sequenza di controllo invocante (il processore ha eliminato dal chiamante gli argomenti utilizzati). Quelli non entrati in gioco (`lz \u w`) sono rimasti nel documento (fra questi la sequenza di controllo `\u` viene a sua volta rimpiazzata dal testo di sostituzione della macro omonima, riconoscibile dagli asterischi.) e alla loro sinistra avranno il testo di rimpiazzamento di `\xyz`. La chiamata `\xyz m \np llz \u w` ⇒ -m-32 a 57-l-lz *bst 21*w

- Nei due casi visti è venuta a mancare quella corrispondenza fra parametri e argomenti con cui in genere vengono pensate questo tipo di macro che nella mancata corrispondenza, se non finalizzata a obiettivi specifici come in 2.4.3, 2.6.2.2 verrebbero a perdere la logica che le sostiene. Il controllo circa questa corrispondenza si ottiene facilmente creando con le graffe un gruppo per ogni parametro. Oppure si può agire con i delimitatori. Una corrispondenza da graffe: `\xyz{m \np}{llz}{\u w}` restituisce -m 32 a 57-llz-*bst 21*w-

- Sono date la macro `\def\foo#1{\count258=4#1\relax}` e la chiamata `\foo{23}`. In questo caso l'argomento è il numero 23 che, scritto in coda al numero 4, assegna al contatore `\count258` il numero 423[3]. Infatti `\number\count258` = 423. In questo caso l'invocazione `\foo{23}` è stata

[3]Facendo attenzione a non utilizzare \count0, il contatore di pagine dei documenti LaTeX, si è scelto il contatore 258 che, essendo utilizzabile, indica la disponibilità dell'estensione ε-TEX con i suoi 32768 contatori, espressi dai naturali da 0 e $2^{15} - 1$.

fatta preliminarmente per passare a `\count258` il valore richiesto e per via della particolare formulazione del *testo* di sostituzione della macro. Infatti il *testo* è una espressione o meglio una uguaglianza aritmetica che la chiamata `\foo{23}` non può visualizzare direttamente; lo si può fare con `\number\foo{23}` che mostra in termini aritmetici il testo di sostituzione della macro: 423=423.

2.3 Macro, parametri e delimitatori

2.3.1 Da *Il TEX*

I parametri di una macro possono essere utilmente delimitati.
In `\def\DoASentence#1#2.{{#1#2}}` il secondo parametro è delimitato da un punto per cui il primo argomento è il primo carattere o s.c. mentre il secondo argomento va dal secondo token fino a quello immediatamente seguito dal punto, gli spazi anche in questo caso non sono considerati. E nella chiamata `\DoASentence \bf Questo è l'argomento.` si ha che `1# <- \bf` e `2# <- Questo \ ‘e l'argomento` notando che il punto, necessariamente presente come delimitatore del secondo argomento non ne fa parte e non appare nel testo di sostituzione. `\DoASentence \bf Questo è l'argomento.` ⇒ **Questo è l'argomento**

La seconda coppia di graffe del testo di sostituzione ha lo scopo di delimitare l'effetto di `\bf`. Con `\def\DoASentence#1#2.{#1#2}`, dovremmo chiudere la chiamata reimpostando `\rm`.
`\DoASentence \bf Questo è l'argomento.\rm` ⇒ **Questo è l'argomento**

2.3.2 Da *TEX in practice*

In `\def\sezione#1 #2\par{\medskip\noindent{\bf#1 #2\par}}` entrambi i parametri sono delimitati, da uno spazio e da `\par` che, in quanto delimitatori , sono presenti anche nel chiamante ma non saranno trasportati nel testo di sostituzione. Il primo argomento della chiamata (vedi penultima riga del paragrafo) è ciò che precede lo spazio, cioè `3.4.`. Lo spazio è presente anche tra i parametri nel testo di sostituzione e separa il testo dei due argomenti (questo spazio è conservato). Il secondo argomento è `Parametro di testo` delimitato da `\par`. Il `\par` del testo sostituzione serve ad evidenziare l'indicazione di Sezione staccandola dal testo che la segue, che forma un nuovo paragrafo. Infatti la chiamata `\sezione \sezione 3.4. Parametro di testo\par` ⇒ **3.4. Parametro di testo**

Che facciamo seguire da un testo che evidenzia l'effetto `\par`.

2.4 Macro con parametro e annidate

Macro esterna ed interna hanno ciascuna un parametro che in entrambe non potrà che essere indicato con il numero 1. Poiché il parametro della macro

esterna sarà sostituito dal suo argomento durante la prima espansione e quello della macro interna nella seconda, TEX li distingue indicando quello attuale con `#1` e quello che lo diventerà alla seconda espansione con `##1`.

2.4.1 Da *Appunti di programmazione in LaTeX e TeX*

è data la macro `\def\abc#1{\def\xyz##1{##1hkj#1}}`: in `\abc` è annidata la macro `\xyz`. Il parametro della macro esterna `\abc` è indicato da `#1`, quello della macro interna con `##1`. Nella invocazione `\abc{MNP}`, poiché stiamo trattando con la macro che ha il primo parametro, il primo passo sarà la sostituzione del parametro `#1` con l'argomento `MNP` dopo di che la sequenza di controllo `\abc` è sostituita dal testo di rimpiazzamento della macro omonima ottenendo la macro `\def\xyz#1{#1hkjMNP}}` in cui TEX ha trasformato `##1` in `#1` e che rimane in memoria, in quanto macro.

La seconda invocazione, `\xyz{43}`, assegna l'argomento al nuovo parametro `#1` della macro invocata e questa sostituisce la sequenza di controllo del chiamante col proprio testo di sostituzione ⇒ 43hkjMNP

▷ `\def\abc#1{\def\xyz##1{##1hkj#1}}`
▷ `\abc{MNP}` ⇒ `\def\xyz#1{#1hkjMNP}}`
▷ `\xyz{43}` ⇒ 43hkjMNP

2.4.2 Idea da *TeX in practice*

Si abbia la macro `\LetterHead` con tre parametri per stampare nome, via e città di chi scrive una lettera.

```
\def\LetterHead #1#2#3{%
   \leftline{\textsf{#1}}
   \leftline{\textsf{#2}}
   \leftline{\textsf{#3}}
   }
```

`\leftline` è una macro TEX la cui espansione utilizza la macro TEX `\line` anch'essa da espandere. Pur trattandosi di macro TEX se ne riportano le definizioni.
`\def\leftline#1{\line{#1\hss}}`
`\def\line{\hbox to\hsize}`
`\hss`: spazio glue orizzontale infinitamente allungabile, spinge l'argomento `#1` a sinistra.
Guardando alla chiamata del solo argomento `#1`, `\LetterHead{Pinco Pallino}` diventa prima `\leftline{\textsf{Pinco Pallino}}` ⇒ `line{\textsf{Pinco Pallino}\hss}` ⇒ `\hbox to \hsize{\textsf{Pinco Pallino}\hss}` ⇒ Pinco Pallino

Mentre la chiamata con i tre argomenti `\LetterHead{Pinco Pallino} {viale dei Sogni 15} {55878 Chissà Dove}` restituisce

Pinco Pallino

viale dei Sogni 15
55878 Chissà Dove

Per aggiungere a `\LetterHead` una macro per firma e titolo si vengono ad avere due macro annidate `\def\LetterHead #1#2#3{...\def\Signature ##1{\leftline{Cordiali saluti}\leftline{##1 #1}}}` che indicano il parametro della macro esterna con `#1`, per l'argomento *firma*, che viene inserito alla prima espansione e quello della macro interna con `##1` per il *titolo* inserito alla seconda espansione dopo che TEX avrà trasformato `##1` → `#1`. La macro diventa:

```
\def\LetterHead #1#2#3{%
   \leftline{\textsf{#1}}
   \leftline{\textsf{#2}}
   \leftline{\textsf{#3}}
   \par
   $\vdots$
    \par
   \def\Signature##1{%
     \leftline{Cordiali saluti}
     \leftline{##1 #1}
       }}
```

E con le chiamate:
`\LetterHead{Pinco Pallino}{viale dei Sogni 15}{55878 Chissà Dove}`
e
`\Signature{Il mago}` si ottiene

Pinco Pallino
viale dei Sogni 15
55878 Chissà Dove
⋮
Cordiali saluti
Il mago Pinco Pallino

2.4.3 Da *Il TEX* : Macro che invoca macro, parametri e delimitatore

Si tratta di un esempio basato su due macro e una chiamata che mostra come TEX gestisce argomenti con il delimitatore spazio:
▷ `\def\thm#1 {\sezione Teorema\kern.5em#1}`
▷ `\def\sezione#1 #2\par{\par\medskip\noindent\textbf{#1\kern .5em}\textsl{#2}\par}`.
La chiamata `\thm5.7 Ogni spazio di Hilbert è riflessivo.\par` restituisce:

Teorema 5.7 *Ogni spazio di Hilbert è riflessivo.*

Vediamo come vanno le cose.
Poiché il parametro della macro `\thm` è delimitato da uno spazio così come nella chiamata lo sono i token di `5.7` questi divengono l'argomento per la macro `\thm`. L'assegnazione dell'argomento al suo parametro nel testo di sostituzione precede l'espansione di `\thm`. E quando la sequenza di controllo `\thm` del chiamante è sostituita dal testo di rimpiazzamento (il restante del testo del chiamante è rimasto in loco) si ottiene `\sezione Teorema\kern.5em5.7 Ogni spazio di Hilbert è riflessivo.\par` che TEX vede come una chiamata alla macro `\sezione` in cui il primo parametro è delimitato da uno spazio, proprio come la prima sequenza argomento del chiamante è delimitata dallo spazio che segue 5.7 mentre il secondo è delimitato da `\par`. Per cui da un lato con `#1` ← `Teorema\kern.5em5.7` e `#2` ← `Ogni spazio di Hilbert è riflessivo.` diventano primo e secondo argomento della macro `\sezione` e dall'altro segue che `\sezione`, essendo la sequenza di controllo di un chiamante, sarà sostituita dal testo di rimpiazzamento della macro omonima ottenendo la sequenza `\par\medskip\noindent\textbf {Teorema\kern.5em5.7}\kern.5em\textsl{Ogni spazio di Hilbert è riflessivo}.\par`. Che lasciata espandere restituisce:

Teorema 5.7 *Ogni spazio di Hilbert è riflessivo.*

Niente male, come un *puzzle*; ma una macro senza delimitatori e con gli argomenti in graffe è una situazione più tranquilla.

2.5 \edef e le macro

Il processore di input memorizza una macro del tipo `\def\abc{nrtd45tx}` di un documento TEX sotto forma di lista di token riferibile alla sequenza di controllo `\abc`. La successiva digitazione della s.c. `\abc` è trattata dal processore di espansione come una chiamata per la macro che a seguito di questa invocazione viene espansa sostituendo la sequenza `\abc` del chiamante con il proprio testo di rimpiazzamento che appare come "valore" della sequenza. Se nel testo di sostituzione oltre ai caratteri sono presenti parametri e/o altre sequenze di controllo, l'espansione della macro sarà sensibile a questi elementi per cui l'esito dell'espansione dipenderà dal valore che essi avranno nel momento in cui la macro sarà invocata (vedi `\foo` in 2.2.1).

Diverso è il caso della macro definita dalla primitiva `\edef`: i processori non si limitano a memorizzare la lista dei token e procedono alla espansione di quanto c'è di espandibile nel testo di sostituzione ottenendo una nuova (non solo formalmente) definizione della macro. A questo punto la `\edef` ha esaurito il suo compito e il nuovo testo di sostituzione definisce una macro di tipo `\def`. (vedi in particolare 2.6.1 e 2.6.2).

Per questa sua capacità di espansione diretta , la macro definita da `\edef` è sensibile ai valori del suo testo di sostituzione al momento in cui è letto dal processore e digitare la sua sequenza di controllo significa soltanto conoscere l'esito di un'espansione avvenuta in precedenza.

2.5.1 Da *TEX in practice*

1. Consideriamo le macro:
   ▷ `\def\xx{ABC}`
   ▷ `\def\yy{\xx}`
   ▷ `\edef\zz{\yy}`

   Il processore di input interviene sulle tre macro, quello di espansione espande il testo di sostituzione di `\edef`. Per cui `\edef` espande `\yy` che diventa una chiamata per la seconda macro che si espande in `\xx` che viene rimpiazzata dal suo testo di sostituzione `ABC`: `\edef\zz{\yy}` ⇒ `\def\zz{ABC}`. Pertanto `\zz` ⇒ ABC.

   Se effettuiamo l'assegnazione `\def\xx{345}` vedremo che mentre le macro definita da `\def` assumono il valore attuale, quella definita da `\edef` rimane col valore assunto al momento della definizione. Si ottengono infatti:
   `\xx` = 345
   `\yy` = 345
   `\zz` = ABC

2. Sono definite le macro
   `\def\xx{Fun}`: (se chiamata, `\xx` ⇐ `Fun`)
   `\edef\zz{\xx}`: (espande `\xx`: `\xx` ⇐ `Fun`; `\edef\zz{\xx}` ⇒`\def\zz{Fun}`)
   `\edef\aa{\noexpand\xx}`: (Comnoexpand inibisce l'espansione di `\xx` che però dalla precedente macro ha assunto il valore Fun)

   In questo caso `\edef\aa{\noexpand\xx}` ⇒ `\def\aa{\xx}` per cui la chiamata `\aa` restituisce il valore finale dell'espansione del il testo di sostituzione.
   `\aa` = Fun
   `\xx` = Fun
   `\zz` = Fun

   Ancora:
   ▷ `\def\bb{\xx}` (se chiamata, `\bb` vale `\xx` che vale Fun)
   ▷ `\edef\qq{\noexpand\xx\xx}` (il primo `\xx` non viene espanso, ma `\edef` gli fa assumere il valore assunto in precedenza, cosa che `\def` non consente, il secondo si espande e va a prendersi il valore Fun)
   ▷ `\def\pp{\noexpand\xx\xx}` (`\pp` si espande al momento della chiamata andando a sostituire la sequenza di controllo del chiamante che diventa `\noexpand\xx\xx`. Il primo `\xx` non può espandersi (rimanendo del tutto inattivo), lo fa soltanto il secondo: la macro `\xx` è chiamata una sola volta.)

   Invocando le macro si ha:
   `\qq` = FunFun
   `\bb` = Fun
   `\pp` = Fun

3. Confronto `\edef` `\def`
   Le macro:

```
\def\nome{Mario}
\def\nomea{\nome}
\edef\nomeb{\nome}
\edef\nomec{\noexpand\nome}
\edef\nomed{\noexpand\nome\space \nome}
\def\nome{Giovanni}
```

Si invocano per espanderle le macro definite con `\def` e per conoscerne il valore assunto per quelle definite da `\edef`. L'ultimo nome associato alla macro `\nome` è Giovanni, il primo Mario. Seguono cinque chiamate.

▷`(\def)\nome` ⇒ *Giovanni.* Il testo di sostituzione è quello attuale (l'ultima assegnazione).
▷`(\def)\nomea` ⇒ *Giovanni.* `\nomea` ⇐ `\nome` ⇐ Giovanni.
▷`(\edef)\nomeb` ⇒ *Mario.* `\edef` fa espandere `\nome` al momento della definizione, il nome Giovanni non è stato ancora assegnato, per cui `\nome` ⇐ Mario.
▷`(\edef)\nomec` ⇒ *Giovanni.* `\noexpand` non fa espandere `nome`, l'espansione di questo `\edef` avviene quando la macro `\nome` è stata già stata sostituita dal nome Giovanni. Per cui `\nomec` ⇐ Giovanni.
▷`(\edef)\nomed` ⇒ *Giovanni Mario.* Nell'argomento di questa `\edef` è presente `\noexpand\nome` che, come visto, dà Giovanni; il secondo viene espanso in fase di definizione quando il nome attuale è ancora Mario.

4. Gruppo e le espansioni ripetute per annidare

```
\def\xx{Inclusioni:}
\centerline{\xx}
{
\def\xx{$\bullet$ }
\edef\temp{\xx}
\edef\temp{ABC \temp DEF }
\edef\temp{XX \temp YY }
\edef\temp{123 \temp 456}


\centerline{\temp}
}
```

Con `\temp` si definisce una macro che controllata da `\edef` assume come primo valore `\bullet`. Il secondo `\edef` definisce una stringa con al centro il precedente `\bullet` a sinistra ABC e a destra DEF e la cosa si ripete altre due volte caricando al centro il vecchio che è annidato nel nuovo. I vari passaggi formano una unità gruppale. L'ultimo `\temp` invocato all'interno del gruppo mostra come è andata.

Inclusioni:
123 XX ABC • DEF YY 456

2.5.2 Da *The TEXbook,* raddoppiare

1. `\def\double#1{#1#1}`: `\double` raddoppia l'argomento che riceve
   `\edef\a{\double{xy}}`: espande `\double` e assume due volte il testo dell'argomento
   `\a` = xyxy
   `\edef\a{\double\a}` : espande `\double` e assume due volte il valore attuale di `a`
   `\a` = xyxyxyxy
   `\edef\a{\double\a}` espande `\double` e assume due volte il valore attuale di `a`
   `\a` = xyxyxyxyxyxyxyxy

2. `\def\n{4}`
   Si prepara a definire `\asts` con 1/2/3/4/5 asterischi:
   ▷ `\edef\asts{\ifcase\n\or*\or**\or***\or****\or*****\fi}`
   `\asts\asts\asts\asts\asts\asts` = **********

2.5.3 Da *TEX by Topic,* modi

Un'applicazione particolare di `\edef` è la macro `\modedef` che, inserita in una certa situazione del documento, invocata ci informa sul 'modo' di quell'ambiente.

▷ `\edef\modedef{Macro definita in '\ifvmode vertical\else \ifmmode math\else horizontal\fi\fi' mode}`

La macro `\modedef` controlla la modalità in cui si trova TEX nel contesto in cui essa è inserita. In questo caso poiché viene scritta di seguito a questo testo la modalità attuale è: Macro definita in 'horizontal' mode

Qui la macro viene riscritta in un apposito paragrafo e la risposta sarà: Macro definita in 'vertical' mode

Qui la macro è riscritta in ambiente matematico e per rendere leggibile la risposta la chiamata va proposta come `\mbox{\modedef}`.
Macro definita in 'math' mode

2.5.4 Macro e prefissi

Il significato della definizione di una macro può essere cambiato dalla presenza dei prefissi: `\global`, `\long`, `\outer`.

1. Una assegnazione ad una macro definita in un gruppo esplicito o implicito è locale. Da
   `\def\pippo{abc}`
   `{\def\pippo{xyz}\pippo}\pippo` otteremo: xyzabc. L'assegnazione interna al gruppo esplicito non ha alcun effetto al suo esterno.

Se la macro interna al gruppo viene definita come globale , `\global\pippo` oppure `\gdef\pippo`, allora l'assegnazione `xyz` avvenuta internamente al gruppo si estende anche fuori di esso sostituendosi all'assegnazione preesistente. (Da *Appunti di programmazione*.)

Il prefisso `\global` può essere applicato anche a `\edef` che in questo caso diventa `\xdef`.

2. Il prefisso `\long` segnala che gli argomenti di una macro possono contenere il token `\par` o un salto di riga. Data la macro `\long\def\a#1{#1}` e l'assegnazione `\a{Riga uno\par Riga due\par Riga tre}` si ottiene: Riga uno

   Riga due

   Riga tre

3. Il prefisso `\outer` segnala una macro non utilizzabile in un testo di sostituzione di un'altra macro, in un argomento, nel preambolo di un allineamento.

   `\beginsection` e `\proclaim` sono esempi di macro che *The TEXbook* a pag. 355 definisce come `\outer`.

2.6 \noexpand e \expandafter

2.6.1 Da *TEX by Topic*

Esempio in cui si guarda alla primitiva `\noexpand` che nel contesto di una `\edef` inibisce l'espansione del primo token che la segue. Due casi non scontatissimi.

1. Le macro sono `\def\a{\b} \def\b{c} \def\d{\e} \def\e{f}` e `\edef \g{\expandafter\noexpand\a \d}`. Guardando all'espansione prodotta da `\edef`: `\expandafter` salta `\noexpand` e interviene su `\a` con l'espansione di testo sostituendola con `\b` (un solo passo di espansione) mentre `\d`, controllata da `\edef`, si espande completamente in `f` per cui la macro diventa `\edef\g{\noexpand\b f}`, e qui `\edef` non potendo espandere direttamente `\b` ha terminato il suo iter e la macro diventa `\def\g{\b f}`. Quando poi `\g` viene invocata l'espansione si completa. `\g` =cf

2. Scrivendo `\edef\next{\def\thing{text}}` pensiamo di chiedere a `\edef` di espandere la macro `\thing`, ma TEX non può farlo in quanto quella macro non è ancora definita. Occorre correggerci scrivendo `\edef\next{\def \noexpand\thing{text}}`: in questo caso `\noexpand` inibisce l'espansione di `\thing` e, visto che `\edef` ha terminato il suo compito, la macro diventa `\next\def\thing{text}` dove il significato di `\next` è la definizione della macro `\def\thing{text}`. Per cui l'invocazione della macro `\thing` dovrà essere preceduta dalla sua messa a disposizione con un `\next`. Ora `\thing` = "text".

2.6.2 Da *TEX in practice*: `\noexpand` e `\expandafter`

1. `\def` e `\expandafter`

   ▷ `\def\xx [#1]{L'#1 sono io. Il testo di sostituzione è sempre #1}`
   ▷ `\def\yy{[ABC]}`

   La chiamata alla prima macro va fatta ponendo l'argomento tra quadre. Facendola nella forma `\xx\yy`, senza che la sequenza `\xx` sia seguita dalla parentesi quadra aperta che il processore si aspetta, il suo sviluppo si blocca.

   Le due chiamate possono essere disposte una dopo l'altra a condizione che siano nella forma `\expandafter\xx\yy`. Infatti `\expandafter` salva senza espanderla la sequenza di controllo `\xx` mentre espande `\yy`. Questa è sostituita dal suo testo di rimpiazzamento `[ABC]` ottenendo `\xx[ABC]` dove `ABC` è diventato l'argomento e va a prendere il suo posto nel testo della macro `\xx` che ora sostituisce la sequenza di controllo che l'ha invocata con il suo testo di rimpiazzamento.
   Il risultato: L'ABC sono io. Il testo di sostituzione è sempre ABC

2. `\edef` e `\expandafter`

   ▷ `\def\xx[#1]{XX ``#1'' YY}`
   ▷ `\def\yy{[ABC]}`
   ▷ `\edef\zz{\expandafter\xx\yy}`

   Il motore è la primitiva `\edef` che espande completamente il suo testo di rimpiazzamento: `\expandafter\xx\yy`. `\expandafter` mette in attesa `\xx` ed espande `\yy`: `\expandafter\xx\yy` ⇒ `\xx[ABC]` che invoca la macro `\xx` passandogli l'argomento (che dovendo essere proposto tra quadre è) ABC. Infine la macro `\xx` rimpiazza la sequenza di controllo con il suo testo di sostituzione che è: XX "ABC" YY

2.6.3 Dal *The TEXbook*, un classico.

`\uppercase\expandafter{\romannumeral\n}`, con n=2011. L'argomento di `\uppercase` deve essere un testo minuscolo tra graffe. `\expandafter` salta la `{` permettendo a `\romannumeral\n` di creare una stringa alfabetica in caratteri minuscoli esprimente il numero assegnato chiusa tra graffe. Quello che si aspetta `\uppercase`, che restituisce quei caratteri in maiuscolo: MMXI.

2.7 `\ifx`

`\ifx` confronta il significato dei due token che lo seguono. Se si tratta di token complessi si confrontano i loro significati, come negli esempi che seguono, dove il primo token è l'argomento di una macro mentre il secondo la prima volta è `\relax` e la seconda `\@empty`.

2.7.1 Da *LaTeX* di Claudio Beccari

Due applicazioni:

1. ```
   \def\a{Vero}
   \let\b=\relax
   \def\confronta#1{\ifx#1\relax vero\else falso \fi}

   \confronta{\a} = falso
   \confronta{\b} = vero
   ```

2. ```
   \def\b{}
   \makeatletter
   \def\xxx#1{\ifx#1\@empty vero\else falso\fi}
   \makeatother
   \xxx\a = falso
   \xxx\b = vero
   ```

2.8 \futurelet

`\futurelet` $\langle token_1 \rangle \langle token_2 \rangle \langle token_3 \rangle$ fa tre mosse:

▷ `\let`$\langle token_1 \rangle$ = $\langle token_3 \rangle$, con $\langle token_1 \rangle$ sequenza controllo di una macro o comando tipo `\next`

▷ rimuove $\langle token_1 \rangle$ dalla lista dei token

▷ espande $\langle token_2 \rangle$ che in genere è una macro.

I due prossimi esempi hanno la stessa struttura logica ma le rispettive macro con il condizionale `\ifx` sono abbastanza diverse. In entrambi i casi le macro sanno riconoscere la presenza di un particolare indicatore (`[`, `*`) che porterà a saper scegliere tra due diversi comportamenti. La prima dopo aver scelto l'opzione affida la sequenza di controllo della macro da attivare a `\next`, mentre TEX rimuove tutto il codice della macro condizionale tranne il `\next` finale, che farà partire l'ultima macro. La macro condizionale del secondo esempio attiva l'ultima macro con `\expandafter`.

2.8.1 Esempio da *TeX in practice*

Si definiscono quattro macro.

```
\def\xxWithOpt [#1]#2{\textbf{\textsl{#1 + #2}}}
\def\xxNoOpt #1{\textsc{#1}}
\def\xx{\futurelet\xxLookedAtToken\xxDecide}
\def\xxDecide{%
     \ifx\xxLokedAtToken [%
     \let\next=\xxWithOpt
     \else
```

```
\let\next=\xxNoOpt
\fi
\next}
```

Se la chiamata è `\xx[a]{b}`, con `a` argomento opzionale e `b` obbligatorio, il processore di input restituisce i token, `\xx•[•a•]•{•b•}`. Il primo token (la s.c. `\xx`) chiama la macro omonima ed è sostituito dal suo testo di rimpiazzamento creando la lista Comfuturelet•\xxLookedAtToken•\xxDecide• [•a•]•{•b•}. La primitiva `\futurelet` assegna al primo token che lo segue una copia del terzo token (cioè pone `\let\xxLookedAtToken=[`) e il processore rimuove dalla lista in quanto utilizzati sia `\futurelet` sia `\xxLookedAtToken` e la lista diventa `\xxDecide•[•a•]•{•b•}`. Si espande il suo primo token (la s.c.) sostituito dal testo di rimpiazzamento della macro `\xxDecide` in cui, essendo verificata la prima alternativa, viene assegnato a `\next` il valore `\xxWithOpt` mentre il resto del codice di `\ifx` che ha svolto il suo compito viene rimosso. Della macro rimane soltanto l'ultimo `\next` che nel nostro caso vale `\xxWithOpt` per cui la lista che diviene `\xxWithOpt•[•a•]•{•b•}`, con la sua s.c. invoca la macro omonima a cui prima passa gli argomenti avendone poi la sequenza di controllo sostituita dal testo di rimpiazzameto, argomenti inclusi: `\textbf{\textsl{a + b}}` ⇒ ***a + b***.

Con la chiamata `\xx{o}` la sequenza di controllo `\xx` è rimpiazzata dal testo di sostituzione e si ha `\futurelet•\xxLookedAtToken•\xxDecide•{•o•}`. `\futurelet` pone `\let\xxLookedAtToken={` per cui `\next=\xxNoOpt` e si ha `\xxNoOpt•{•o•}` che invoca la macro omonima ottenendo `\textsc{o}` ⇒ o.

2.8.2 Esempio da *Appunti di programmazione in LaTeX e TeX*

Il processo con argomento opzionale è a due scelte come lo è quello in cui si hanno due forme alternative che si distinguono per un asterisco. Nel caso di `\mybox{abc}` si ottiene un filetto attorno al box del testo `abc`, in quello di `\mybox*{abc}` il font del testo è *sans serif*. Questo esempio, se escludiamo l'uso di `\expandafter`, si sviluppa con gli stessi passi del caso precedente. Quattro definizioni, in cui `\futurelet` è uno snodo fondamentale, bastano a far corrispondere alle due invocazioni il risultato previsto.

Le definizioni:

```
\def\@mybox#1{\fbox{#1}}
\def\@myboxs#1#2{\fbox{\ttfamily #2}}
\def\mybox{\futurelet\next\@myboxchoose}
\def\@myboxchoose{%
    \ifx\next*
    \expandafter\@myboxs
    \else
    \expandafter\@mybox
   \fi}
```

La chiamata `\mybox{abc}` attiva l'omonima definizione che con il suo testo di rimpiazzamento sostituisce la sequenza di controllo `\mybox` della macro invocan-

te ottenendo la lista di token: \futurelet•\next•\@myboxchoose•{•a•b•c•}. Il processore attiva, in quanto primo token della lista, \futurelet che assegna al primo token che lo segue una copia del terzo (cioè pone \let\next={) che rimane in lista dalla quale invece rimuove sia \futurelet che \next in quanto utilizzati. Della lista attuale che è \@myboxchoose•{•a•b•c•} il processore attiva l'espansione di \@myboxchoose, primo token, sostituendolo col testo di rimpiazzamento della macro che, essendogli noto che \next≠*, è dato dal testo che segue \else creando la lista \expandafter•\@mybox•\fi•{•a•b•c•}. Ora \expandafter espande \fi che, come \else, ha espansione vuota e nella lista rimane \@mybox{abc} che a sua volta si espande sostituito dal suo testo di rimpiazzamento, \fbox{abc}, con cui costruisce un box filettato attorno all'argomento: → \mybox{abc} = abc.

Con l'invocazione \mybox*{abc} la lista \futurelet•\next•\@myboxchoose •*•{•a•b•c•} ha un token in più, l'asterisco, con cui \futurelet, con \let\next= *, fa diventare vero lo \ifx di @myboxchoose e la lista dei token si riduce a \@myboxs*{abc}. Per cui \mybox*{abc} = abc.

2.8.3 Esempio da *TEX in practice*

Chiamando una macro che può avere oltre a quello ordinario un argomento opzionale si può disporre che l'argomento opzionale, se non assegnato, sia comunque visibile sotto la forma di un valore di default.

```
\makeatletter
\long\def\DoLongFutureLet #1#2#3#4{%
 \def\@FutureLetDecide{%
   #1#2\@FutureLetToken
  \def\@FutureLetNext{#3}
 \else
  \def\@FutureLetNext{#4}
 \fi
 \@FutureLetNext}
 \futurelet\@FutureLetToken\@FutureLetDecide}
 \makeatother
```

Per scegliere con \ifx la condizione idonea si utilizza \futurelet. Qui si preferisce invocare la macro \DoLongFutureLet definita in precedenza (e qui richiamata) che mette subito in chiaro gli esiti delle due alternative e che può includere il comando \par. Anche qui sono date quattro macro.

```
\makeatletter

\def\OptArgX #1#2#{%
    \let\@OptArgXTemp = #1%
    \def\@OptArgXDefault{[#2]}
    \DoLongFutureLet{\ifx}{[}{\@OptArgXTemp}{\@OptArgXB}
  }
\def\@OptArgXB{\expandafter\@OptArgXTemp\@OptArgXDefault}
```

```
\def\xx{\OptArgX{\@xx}{Opt}}
\def\@xx [#1]#2{%
  la chiamata restituisce l'argomento ordinarioğ\\ ğ ‘‘\textit{#2}’’ e
  opzionale ‘‘\textit {#1}.’
 }

\makeatother
```

La chiamata `\xx[30pt]{DEF}` invocando la macro `\xx` che non ha parametri non impegna i propri argomenti che sono lasciati in loco e si limita a ricevere il testo di rimpiazzamento della macro invocata ottenendo `\OptArgX{\@xx}{Opt}[30pt]{DEF}`. Questa invoca l'omonina macro condizionale passandole gli argomenti: a `\@OptArgXTemp` una copia di `\@xx` e trasferendo in `\@OptArgXDefault` il secondo argomento della lista. Ora `\DoLongFutureLet` ha dinanzi a sé la `[` della lista e attiva l'opzione `\@OptArgXTemp` (un parametro temporaneo tipo `\next`) che vale quanto è rimasto in lista, `\@xx[30pt]{DEF}`. Questa chiamata dopo aver assegnato ai parametri della macro invocata i rispettivi argomenti ha la sequenza di controllo rimpiazzata dal testo di sostituzione della macro che infine ci dice: *la chiamata restituisce l'argomento ordinario "DEF" e opzionale "30pt."* .

La chiamata `\xx{ABC}` ci dà `\OptArgX {\@xx}{Opt}{ABC}` che attiva `\@Opt ArgXTemp` assegnando, come la lista indica, a `\@OptArgXTemp` una copia di `\@xx` e trasferendo in `\@OptArgXDefault Opt`. Questa volta `\DoLongFutureLet` non trova `[` e attiva `\@OptArgXB` che dopo aver reso attuale il valore di `\@OptArgX Default`, prendendo gli `[Opt]` che riassegna seppur indirettamente a `\xx` come argomento opzionale, attiva `\@OptArgXTemp` che visto come lista può essere scritto `\@xx[Opt]{ABC}` e che infine ci dice: *la chiamata restituisce l'argomento ordinario "ABC" e opzionale "0pt."*

2.9 Esempi `\ifodd` e `\ifnum`

Sia `\ifnum` che `\ifodd` gestiscono registri numerici. La macro `\oggi` è definita nel preambolo con le sequenze di controllo `\two@digits` di LaTeX, `\day`, `\month` e `\year` di TeX. Per l'ora si definisce la macro `\adesso` che usa l'aritmetica di `TeX` e per aggiungere lo zero alla cifra dei minuti minori di dieci usa il registro `\toks256` oppure `\adessoi` in cui allo zero si provvede in maniera simile a quanto fa la definizione di `\two@digits`.

1. I casi con `\ifodd`, se verificati, scelgono il typewriter alternativamente allo *italic* per la stampa di un risultato; nel primo caso il numero otto, nel secondo data e ora, nel terzo un prodotto.

   ```
   \newcount\ore
   \newcount\minuti
   \newcount\min
   \minuti=\time
   \min=\minuti
   ```

```
▷\def\adesso{
       \divide\min by 60
       \ore=\min
       \multiply\min by -60
       \advance\minuti by \min
       \ifnum\minuti<10 \toks256={0} \else \toks256={}\fi
       \space\number\ore$.${\the\toks256}\number\minuti}

▷\def\adessoi{
       \divide\min by 60
       \ore=\min
       \multiply\min by -60
       \advance\minuti by \min
       \space\number\ore$.$\ifnum\minuti<10  0\number\minuti
       \else \number\minuti\fi}

\newcount\nct
\count0 = 19
\nct = 7
•\ifodd\count0\tt\else\it\fi 8\rm  ⇒ 8
•\ifodd\nct\tt\else\it\fi\oggi\adesso\rm  ⇒ 27/08/2011  16:38
•\ifodd\nct\tt\else\it\fi\multiply\nct by 13\the\nct  ⇒ 91
```

2. Le coordinate polari dei vertici di un poligono regolare inscritto nella circonferenza di raggio unitario, centro nell'origine del sistema di riferimento: `\ifnum`, e il ciclo `\loop`, `\repeat`.

```
\newcount\cont
\cont=0
▷ \def\coordpol#1{
    \loop
    \ifnum\cont<360\advance\cont by #1 \space (1,\the\cont)
    \space\repeat
  }
```

Se `\coordpol{36}`:

(1,36) (1,72) (1,108) (1,144) (1,180) (1,216) (1,252) (1,288) (1,324) (1,360)

3 Testo in un box, le modalità

Lo strumento *box* , un rettangolo (oggetto bidimensionale) caratterizzato da tre misure, in quanto contenitore del carattere tipografico è fondamentale per la composizione dei caratteri con TEX. Per i casi come quelli dei primi cinque caratteri alfabetici bastano due misure *height* e *width* ; l'origine delle misure è vertice basso a sinistra del loro rettangolo, e il lato orizzontale per l'origine

individua la linea di base dei caratteri, guida del loro allineamento orizzontale. Per il box dei casi come quelli del sesto e settimo carattere bisogna conoscere anche la terza misura, *depth* . La profondità, nulla per i primi cinque caratteri, qui avrà un valore indicante quanto il box scende al di sotto della linea di base. Molta parte del testo della pagina TEX è la composizione di questi box e di spazi glue.

3.1 Modalità e box

Nel comporre un testo da sinistra a destra e in linee di un paragrafo, TEX si troverà in una delle seguenti sei modalità divise in tre categorie:

1. modalità orizzontale e strettamente orizzontale
2. modalità verticale e verticale interna
3. modalità matematica e di display matematico

In un `\hbox` il testo è disposto in una linea la cui lunghezza è determinata dal testo, la modalità è quella orizzontale ristretta. Gli elementi della modalità orizzontale sono i `caratteri`, `\unhbox`, `\unhcopy`, `\vrule`, `\hskip`, `\hfil`, `\valign`, i vari `whatsit`.

Il testo nella pagina TEX o in uno specifico box verticale è costruito in modalità orizzontale (ordinaria) detta anche modalità paragrafo.

Nella composizione della pagina la modalità di un testo cambia frequentemente, se si è in modalità verticale e il processore incontra un carattere o un comando orizzontale, TEX incomincia a costruire un paragrafo ponendosi in modalità orizzontale ordinaria, assembla una lista orizzontale di elementi (box, spazi glue, comandi orizzontali) componendoli uno dopo l'altro da sinistra a destra. Incontrando un comando verticale o alla fine del box il paragrafo viene diviso in linee con la lunghezza dello `\hsize` corrente e impacchettati come box strettamente orizzontali che sono "impilati" secondo la modalità verticale interna. Comandi del modo verticale sono `\unvbox`, `\unvcopy`, `\hrule`, `\vskip`, `\halign`.

Vediamo qualche semplice esempio con `\hbox`, `\vbox`, `\vtop`.

- In un box orizzontale il testo è disposto in una stessa linea, la sua lunghezza è quella del testo, la modalità strettamente orizzontale . Oltre ai caratteri possono essere immessi e comandi come `\chardef`, `\vrule`, `\hfil`, `\valign`.

  è possibile predeterminare l'ampiezza del box.
  Con `\hbox to 4cm{\it \small(Uno\hfil\vrule\kern2pt\chardef \percento = '\% 7\percento\kern2pt\vrule\hfil Due)}` si ha:
  (Uno | 7%| Due)

- Se la lunghezza del testo di un `\hbox to` supera quella dello `\hbox` e dello `\hsize` della pagina rimane la modalità strettamente orizzontale, allineamento al margine a sinistra della pagina e sforamento a destra. Con `\hbox to 6cm { \small { \it` Se la lunghezza ... orizzontale`}}` si ha:

Se la lunghezza del testo di un `\hbox` *supera quella corrente della riga rimane in modalità strettamente orizzontale*

- `\setbox256\hbox{\vbox{\it\small``` Un `\vbox` all'inizio del gruppo strettamente orizzontale di `\hbox` crea una modalità verticale interna in cui il testo sarà elaborato come lista orizzontale e poi suddiviso in righe con l'ampiezza prevista dallo `\hsize` della pagina.`''}}` restituisce il paragrafo virgolettato. In effetti poiché `\hbox` è preceduto dal comando `\setbox256` che inizializza il registro box 256, TEX salva il testo del paragrafo in questo registro. Per recuperarlo si può scegliere il comando `\box256` che crea l'output e vuota il registro o `\copy256` che porta in output una copia del contenuto del registro. Non c'è necessità di conservare e si sceglie `\box256`.

"Un `\vbox` *all'inizio del gruppo strettamente orizzontale di* `\hbox` *crea una modalità verticale interna in cui il testo sarà elaborato come lista orizzontale e suddiviso in righe con l'ampiezza prevista dallo* `\hsize` *della pagina."*

Andando nel file `log` a scorrere il `\tracingcommands` relativo al paragrafo qui sopra si legge: vertical mode - `\hbox` restricted horizontal mode - `\vbox` internal vertical mode - the character ` horizontal mode - ... - end group character vertical mode.

- Con `\hbox{\hsize=5cm \parindent = 2em \vtop { ... } \hskip5.7 em minus 4em \vtop { ... }}` si hanno due `\vtop` affiancati ad occupare, anche grazie alla spaziatura elastica, tutta l'ampiezza della pagina. Qualcosa di simile a due `minipage` affiancate (ampiezza 5cm) di LATEX.

Orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali.

Orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali.

Nuovo pragrafo orizzontali orizzontali orizzontali

- Un testo può essere immesso in un box verticale con una larghezza predeterminata come nel caso di `\vbox {\hsize=6cm{\hrule\it\small ``Un testo... ''}}` in cui il testo è impilato secondo linee orizzontali lunghe 6cm (in modalità orizzontale detta anche *paragraph*) disposte una sotto l'altra. L'estensione verticale naturale del box è determinata da quella delle linee di testo.

"Un testo può essere gestito da un box verticale con una larghezza predeterminata. In questo caso `\vbox{\hsize=6cm{\it\small `` ... ''}}` *impila il testo secondo linee orizzontali lunghe 6cm (in modalità orizzontale detta anche paragrafo), una sopra l'altra."*

In questo caso, in cui non dichiara preventivamente una specifica altezza per il box, il comando `\vbox` ha calcolato la propria estensione verticale complessiva dal numero e dall'altezza delle proprie righe e ha fatto in modo che quella più bassa sia delimitata dal margine basso del box. La baseline di questa riga è anche la linea di base del `\vbox`(la profondità del `vbox` è quella degli elementi della riga bassa, se hanno tutti profondità zero anche il `\vbox` ha `depth` = 0). Se `\vbox` visto sopra dichiara una propria estensione verticale che supera quella complessiva delle linee di testo, nel caso in cui `\vbox to 4cm{\hsize=6cm\hrule\it\small‘‘Un testo può ...}`, si può vedere che, per essere delimitato dalla linea di chiusura del box, il suo testo deve scivolare in basso distaccandosi da quello superiore. Con `\vbox to 1cm{\hsize=6cm\hrule \it\small‘‘Un testo pu ò ...}` accade il contrario e il testo del box verrà ad essere sovrapposto dal testo che lo segue: è l'estensione verticale propria del box che, se dichiarata, entra nei calcoli di gestione della pagina.

Il comando `\vtop`, che accosta la prima riga al margine alto del suo box, si caratterizza per avere l'altezza e la linea di base del box date da quelle della prima riga, le altre ne determinano la profondità.

3.1.1 Box e cornici

Per incorniciare un box verticale come quello precedente, <VBOX da inserire>, occorre inserirlo nel gruppo-argomento di un `\hbox` che ne tracci con `\vrule` i lati verticali e che a sua volta sia incluso nel gruppo-argomento in un `\vbox` che tracci con `\hrule` quelli orizzontali. In queste condizioni i lati tracciati da `\vrule` e `\hrule` assumeranno, visualizzandole, le dimensioni verticale e orizzontale degli `\hbox` e `\vbox` che rispettivamente li contengono. E il codice sarà:

```
\vbox{\hrule\hbox{\vrule <VBOX da inserire> \vrule}\hrule}
```

“Un testo può essere gestito da un box verticale con una larghezza predeterminata. In questo caso `\vbox{\hsize=6cm{\it \small‘‘ ... ’’}}` *impila il testo secondo linee orizzontali lunghe 6cm (in modalità orizzontale detta anche paragrafo), una sotto l'altra.”*

ma le linee della filettatura toccano il testo.

La cornice può essere distanziata dal testo con la primitiva `\kern`. Due `\kern5pt` inseriti nello `\hbox` e sensibili alla sua modalità orizzontale ne accrescono la larghezza di 10pt, ed essendo il primo a destra e il secondo a sinistra dei rispettivi `\vrule` determinano anche lo spostamento dei lati verticali ai margini dello `\hbox`. Ma il $+\Delta h$ si riverbera anche sulla `\hsize` del `\vbox` in cui lo `hbox` è incluso con un conseguente pari allungamento delle `\hrule` tracciate. In effetti nel calcolo di Δh entra anche il `width` di `\vrule` per cui il lato orizzontale copre gli spessori dei lati verticali. Il codice è

`\vbox{\hrule\hbox{\vrule\kern5pt <VBOX da inserire>\kern5pt\vrule} \hrule}` che restituisce:

"Un testo può essere gestito da un box verticale con una larghezza predeterminata. In questo caso `\vbox{\hsize=6cm{\it \small`` ... ''}}` *impila il testo secondo linee orizzontali lunghe 6cm (in modalità orizzontale detta anche paragrafo), una sotto l'altra."*

Ma non siamo ancora in grado di spostare correttamente i lati orizzontali. Infatti immettendo, ad esempio, un `\kern2pt` fra il primo `\hrule` e `\hbox` ed un altro subito prima del secondo `\hrule`, e quindi in modalità verticale, si ottiene uno spostamento verticale dei lati orizzontali senza però influire sull'altezza del `\vbox` che, dipendendo da quella dello `\hbox` interno, non si adegua e crea indirettamente discontinuità nella cornice. Si può cogliere questa circostanza per visualizzare, assegnando a `\vrule` la `width4pt`, che lo spessore del lato verticale è tutto interno alla larghezza dei lati orizzontali e che questi, indipendentemente dalla scelta di assegnare il loro spessore sotto forma di `height` o `depth`, rispettano comunque lo spostamento verticale assegnato.

A questo proposito si può anche osservare come un testo successivo va ad allinearsi alla baseline del `\vbox`. Infatti:
`\hbox{\vbox{\hrule height.4pt\kern2pt\hbox{\vrule width4pt\kern5pt \vbox<VBOX da inserire>kern5pt \vrule}\kern2pt\hrule depth4pt}allineamento con baseline del vbox}` restituisce:

"Un testo può essere gestito da un box verticale con una larghezza predeterminata. In questo caso `\vbox{\hsize=6cm{\it \small`` ... ''}}` *dispone il testo in una pila secondo linee orizzontali lunghe 6cm (in modalità orizzontale detta anche paragrafo), una sotto l'altra."* allineamento con baseline del vbox

Questo `vbox` con `\hsize=185.12pt`, somma dei valori di `\hsize` del box di testo inserito, dei due `\kern` orizzontali e delle `width` dei due `\vrule` crea le condizioni per allineare il suo testo, una possibile didascalia, ai margini laterali del `\vbox` incorniciato.

Tornando al `\vbox`, salvandolo in un registro come `\setbox256`, la larghezza del `\vbox` della didascalia potrà essere direttamente assegnata con un `\hsize = \wd256`, come si vedrà in un esempio successivo.

Per variare l'altezza del box verticale esterno sarà necessario inserire nell'argomento di `\hbox` un nuovo `\vbox` con il solo scopo di modificare l'altezza del `\vbox` esterno di quanto previsto, ad esempio 3pt. In questo modo si ottiene la struttura proposta negli esercizi de *The TEXbook*.

`\vbox{\hrule\hbox{\vrule\kern5pt\vbox{\kern3pt<VBOX da inserire>
\kern3pt}\kern5pt\vrule}\hrule}`.

Se il <VBOX da inserire> è quello precedente:

> *"Un testo può essere gestito da un box verticale con una larghezza predeterminata. In questo caso* `\vbox{\hsize=6cm{\it \small`` ... ''}}` *dispone il testo in uno stack secondo linee orizzontali lunghe 6cm (in modalità orizzontale detta anche paragrafo), una sotto l'altra"*

Basandosi sulla struttura del codice per la filettatura del box vista qui sopra si potranno definire macro per soddisfare specifiche richieste. Come la macro `\cornice` che prende come primo argomento l'oggetto da incorniciare e come secondo la distanza (in pt) fra oggetto e cornice.
`\long\def\cornice#1#2{\vbox{\hrule\hbox{\vrule\kern#2pt\vbox{\kern #2pt #1\kern#2pt} \kern#2pt\vrule}\hrule}}`. `\long` per incorniciare più paragrafi.

Se poi si deve incorniciare un breve testo o un simbolo il primo argomento della nuova macro sarà un `\hbox`: `\cornice{\hbox{\textbullet}}{2}` o `\cornice{\hbox{\texttt{\string\kern2pt}}}{2}` ottenendo: • \kern2pt.

Se poi si considerano nello stesso `\hbox` due box verticali (un `\vbox` e un `\vtop`) ciascuno con due `\hbox` di un carattere, si avranno due box verticali affiancati che ci fanno anche guardare al loro funzionamento. Inizialmente la linea di base di riferimento dello `\hbox` (visualizzata dalla punteggiata) è quella successiva all'ultima riga del testo del codice. Il `\vbox` a partire da questa linea immette per prima la riga del `k` e *sotto* di questa la riga dello `h`. In questo modo la linea di base originale, inizialmente di `k`, diventa quella di `h` distanziandosi dal testo che la precede. Il `\vtop` si riferisce alla stessa linea di base del `\vbox` (sono nello stesso `\hbox`) che diventa e rimane la linea di base di `k`, la riga di `h` è creata al di sotto di quella originale. Gli `\hbox` di un box verticale si allineano come elementi della stessa colonna.

```
\hbox to 2cm{\vbox{\cornice{\hbox{k}}{2}\vskip2pt\cornice{\hbox{h}}{2}
    \dotfill\vtop{\cornice{\hbox{k}}{2}\vskip2pt\cornice{\hbox{h}}{2}}}
```

k

h.....k

h

3.2 Testo, box e registri

Consideriamo tre `\hbox` con i registri di riferimento in cui memorizzarli:

- lo `\hbox` che ospita un `\vbox` e un `\vtop` di dimensioni assegnate oltre a un breve testo. `\hbox` salvato nel registro box 256 ;

▷`\setbox256\hbox{\hsize=3.1cm \parindent = 0em \vbox{\small\it orizzontali orizzontali1 orizzontali orizzontali2 orizzontali poggiano} \hskip.5em \vtop{\small\it orizzontali orizzontali1 orizzontali orizzontali2 orizzontali orizzontali}\hskip1em \scriptsize segue testo sulla stessa linea di base ancora ancora ancora ancora}`

- lo `\hbox` che ospita un `\vbox` di dimensione assegnata salvato nel registro box 257;

  ▷`\setbox257\hbox{\hsize=3.1cm \parindent = 0em \vbox{\small\it orizzontali orizzontali orizzontali orizzontali orizzontali poggiano}}`

- lo `\hbox` che ospita un `\vtop` di dimensione assegnata e un breve testo salvato nel registro box 258;

  ▷`\setbox258\hbox{\hsize=3.1cm \parindent = 0em\vtop{\small\it orizzontali orizzontali orizzontali orizzontali orizzontali orizzontali}\hskip.5em \scriptsize segue testo sulla stessa linea di base segue testo sulla stessa linea di base segue testo sulla stessa linea di base ancora ancora ancora ancora}`

Si riportano, affiancate in un `\hbox` di due `\vtop`, le definizioni delle macro `\framehbox` e `\frameunhbox`: la struttura del codice è la stessa introdotta in 3.1.1.

```
\def\framehbox #1#2#3#4{\vbox{\hrule%
               height #1pt%
  \hbox{\vrule width #1pt\kern #2pt%
  \vbox{\kern #2pt%
  \vbox{\hsize #3\noindent #4}%
  \kern #2pt}%
  \kern #2pt\vrule width #1pt}%
  \hrule height0pt depth #1pt}}
```

```
\def\frameunhbox #1#2#3{\vbox{\hrule%
                 height #1pt%
  \hbox{\vrule width #1pt\kern #2pt%
  \vbox{\kern #2pt%
  \vbox{\noindent #3}%
  \kern #2pt}%
  \kern #2pt\vrule width #1pt}%
  \hrule height0pt depth #1pt}}
```

Su queste premesse si considerano tre esempi.

1. Dovendo gestire più vbox, TEX controlla se le loro dimensioni orizzontali sono tali da poterli allineare su una stessa riga proprio come i box dei caratteri di una riga di testo, seguendo la loro linea di base. Allineamento che è ben visibile: il box a sinistra è un `\vbox` e la sua riga bassa definisce la linea di base, quello alla sua destra un `\vtop` ed è la sua prima riga ad appoggiarsi sulla linea di base così come il breve testo che lo segue. `\vbox` e `\vtop` sono contenuti insieme al breve testo come argomento dello stesso `\hbox` che stabilisce con `\hsize=3.1cm` la loro dimensione orizzontale. La modalità è quella orizzontale ristretta, il tutto salvato dal comando `\setbox256` nel registro box 256.

Il comando `\copy256` fa una copia del contenuto del registro e lo stampa nella pagina conservando la modalità strettamente orizzontale. Per evidenziarlo al meglio il box è stato riquadrato con una filettatura che si distacca dal margine del box di 2pt.

Il comando `\framehbox{.1}{2}{\wd256}{\copy256}` ha quattro parametri: il primo per lo spessore della linea, il secondo i pt che distanziano la filettaura dal box, il terzo la larghezza del box 256, il quarto immette una copia del registro 256 conservandone la modalità originaria.

orizzontali orizzontali1
orizzontali orizzontali2
orizzontali poggiano *orizzontali orizzontali1* segue testo sulla stessa linea di base ancora ancora ancora ancora
orizzontali orizzontali2
orizzontali orizzontali

2. Ma non sempre è utile conservare la modalità strettamente orizzontale originaria: per mettere la lista dei dati nella più flessibile modalità paragrafo TEX rende disponibili le primitive `\unhcopy` e `\unhbox`[4]. Qui il comando per l'incorniciatura è `\frameunhbox{.1}{2}{\unhbox256}`[5], l'ampiezza dello hbox è quella della pagina. `\unhbox256` dopo averlo riprodotto vuota il registro box256.

orizzontali orizzontali1
orizzontali orizzontali2
orizzontali poggiano *orizzontali orizzontali1* segue testo sulla stessa linea di base ancora
orizzontali orizzontali2
orizzontali orizzontali
ancora ancora ancora

3. Prendendo i box salvati nei registri 257 e 258 utilizzandoli come oggetti unitari, componendoli e riquadrandoli con `\frameunhbox{.1}{2}{\noindent \unhcopy257\hskip1em \unhcopy257\hskip1em\unhbox258\hskip1em \unhbox257}` si mostra come la modalità orizzontale paragrafo permette la composizione di un testo trattando alla stessa maniera box *piccoli* come quelli dei caratteri tipografici e *grandi* come quelli memorizzati nei registri.

orizzontali orizzontali *orizzontali orizzontali*
orizzontali orizzontali *orizzontali orizzontali*
orizzontali poggiano *orizzontali poggiano* *orizzontali orizzontali* segue testo sulla
orizzontali orizzontali
orizzontali orizzontali
stessa linea di base segue testo sulla stessa linea di base segue testo sulla stessa linea di base ancora
orizzontali orizzontali
orizzontali orizzontali
ancora ancora ancora *orizzontali poggiano*

[4]Si può notare che la riga del breve testo andando a capo crea la nuova linea di base condizionata dalla profondità del `\vtop`.

[5]Le `\def\framehbox` e `\frameunhbox` generalizzano il codice di incorniciatura già visto.

Se si va a guardare come stanno le cose in una pagina in cui si allineano `\parbox` e Progrminipage, l'eredità genetica di TEX in LATEX non può essere più evidente[6].

4 Ambienti, comandi, box e registri

L'utente TEX, LATEX si trova frequentemente nella situazione di predisporre specifiche azioni su un testo, su oggetto come un grafico o una tabella [7]. Nella base LATEX sono disponibili numerose di queste azioni definite come `environment`. Prendendo il caso di `quote`, se in un documento LATEX appare il codice

`\begin{quote}` testo1 `\small` testo2 `\bfseries` testo3 `\end{quote}`

il comando `\begin{quote}` inizia l'*ambiente* con `\begingroup` e il testo avrà il suo sviluppo con una stampa indentata a sinistra e a destra con il testo1 normale, il testo2 `\small` e il testo3 `\small` e in `boldface`; `\end{quote}` termina ogni azione e chiude l'*ambiente* con `\endgroup`[8]. Ma entriamo negli ambienti con gradualità.

TEX Definire un ambiente è previsto in LATEX e non lo è, almeno esplicitamente, in TEX. Ma è a partire da TEX che nasce l'idea di ambiente: un testo, una figura di un documento su cui agire con istruzioni specifiche. Intanto i gruppi. Creare un gruppo significa confinare l'effetto di uno o più comandi in un ambito previsto, il che avviene generalmente tramite coppie di graffe (gruppi espliciti), spesso annidate e che TEX controlla con appositi contatori accettando solo situazioni di parità. Ci sono poi il gruppo di `\begingroup` `\endgroup` e quello implicito di `\bgroup` e `\egroup` (essendo `\let\bgroup={`; `\let\egroup=}` i due comandi sono graffe implicite). Ma l'aspetto interessante è che `\bgroup` `\egroup`, pur graffa equivalenti, non sono conteggiati da TEX per la parità con le graffe e risultano in tal modo indipendenti da queste, mentre `\begingroup` `\endgroup` non hanno legami o riferimenti alle graffe. Pertanto `\bgroup` e `\egroup` possono essere singolarmente presenti nel testo di definizione di una macro. è questa indipendenza che rende operativa l'idea di `ambiente`: *una coppia di macro, una per impostare l'altra per chiudere un'azione su un testo/oggetto.* E quello che trasforma due singole macro in una coppia è

[6] Mi riferisco in particolare al punto 5.1.4 di *Guide to LATEX* che prima di guardare a come funzionano i box di TEX mi sembrava abbastanza incomprensibile. Come per i `\newenvironment` di cui senza TEX non avevo capito la struttura. Circa l'eredità genetica Claudio Beccari che certamente ne sa tantissimo più di me non è d'accordo e scrive: Perché vuoi parlare di eredità genetica; secondo me sarebbe meglio parlare del fatto che Plain TeX, LaTeX, pdfLaTeX, XeTeX, XeLaTeX, luaTeX, luaLaTeX, ConTeXt, (per no parlare di Omega, Lambda, Aleph e altri programmi della distribuzione TeX meno conosciuti) sono tutti costruiti come insiemi di macroistruzioni che si basano tutti sul substrato dei comandi primitivi del programma tex; i comandi primitivi sono la base di tutto; gli altri sembrano linguaggi diversi, ma sono solo collezioni di macroistruzioni e non costituiscono dei veri programmi eseguibili, anche se i vari sistemi operativi fanno in modo che lo sembrino.

[7] Per i comandi vedi il punto 4.2

[8] Vedi *The LATEX 2ε Sources* p. 202

iniziare un gruppo-ambiente nella prima macro, ad esempio, con `\bgroup` che è del tutto indifferente ai confini originali delle macro e terminarlo con `\egroup` nella seconda macro. Il testo e le istruzioni tra `\bgroup` `\egroup` individuano l'oggetto dell'azione delle due macro. Due esempi.

1. ▷ `\def\amb{` Un ambiente che cambia `\bgroup\bf` lo stile del font del `}`
   ▷ `\def\fineamb{`e si propaga `\egroup` ma termina con `\egroup}`
   (il secondo `\egroup` non è una s.c.)

   Le due chiamate intercalate da un testo

   `\amb` TESTO DEL DOCUMENTO `\fineamb`

   sono rimpiazzate dai rispettivi testi di sostituzione e restituiscono:

   Un ambiente che cambia **lo stile del font del TESTO DEL DOCUMENTO e si propaga** ma termina con `\egroup`

   La presenza di `\egroup` nella seconda macro chiude l'ambiente garantendone l'esistenza; ove mancasse, TEX completerebbe l'elaborazione senza segnalare una mancata parità. Occorre precisare che chiudere il gruppo implicito garantisce la terminazione delle operazioni iniziate nel gruppo evitando anche prosecuzioni potenzialmente pericolose.

   Nota Si già è visto nel punto 2.3.1 come una impostazione presente nel codice di una macro possa facilmente trasmettere un effetto non voluto al suo esterno creando almeno potenzialmente problemi, proprio come qui sopra. Vedremo presto che questo tipo di rischio in un *ambiente* LATEX non è più presente.

2. Nel secondo esempio occorre premettere che `\hbox`, comando primitivo TEX, può ricevere dati solo da un gruppo, graffe o `\bgroup` `\egroup`. E dato che il testo di sostituzione delle macro è delimitato da graffe, il gruppo di `\hbox` che inizia nella prima macro e include anche il testo fra le due può essere iniziato e chiuso solo da `\bgroup` `\egroup`. La stessa condizione che si ha nei gruppi delle dichiarazioni di apertura e chiusura di un `\newenvironment` dove l'inclusione del testo interposto e la delimitazione delle istruzioni possono essere risolti anche con un altro ambiente.

   ▷ `\def\iniziabox{\setbox256\hbox\bgroup\bf}`
   ▷ `\def\chiudibox{\egroup Il verme. \unhbox256}`

   La coppia di macro:

   `\iniziabox` Questa è una descrizione mancata di un animale abbastanza curioso, l'armadillo. `\chiudibox` Non della formica.

   è rimpiazzata dal loro testo di sostituzione e restituisce un testo con una distribuzione non del tutto ovvia.
   Il verme. **Questa è una descrizione mancata di un animale abbastanza curioso, l'armadillo.** Non della formica.

Infatti "Il verme" che si trova in `\chiudibox` tra `\egroup` e `\unhbox256` è esterno al gruppo che `\setbox256` memorizza nel registro e rimane in loco precedendo il flusso portato da `\unhbox256` che a sua volta precede "La formica" anch'essa rimasta al suo posto.

LaTeX, dà all'utente la possibilità di definire una propria macro con il comando `\newcommand{\mionome}`. Ma prima di guardare ai `\newcommand` voglio concludere il filo che a partire dalle coppie di macro, considerate come antesignano, porta l'utente LaTeX a poter costruire un proprio *ambiente*. Il fatto nuovo e molto funzionale è che LaTeX lo fa con due distinti interventi che hanno ruoli complementari. Nel preambolo il comando `\newenvironment{nome}` definisce le istruzioni da applicare al *testo/oggetto* distribuite in due gruppi espliciti distinti, uno per le dichiarazioni di apertura l'altro per quelle di chiusura. Nel documento, indipendentemente dalle operazioni previste nelle dichiarazioni, nella forma `\begin{nome}` *testo/oggetto* `\end{nome}`, il comando `\begin{nome}` inizia il gruppo *ambiente* e attiva le operazioni di elaborazione del *testo/oggetto* chiuse da `\end{nome}` che termina il gruppo. Mentre nella coppia di macro TeX è l'utente a sigillare l'azione delle macro all'interno dell'*ambiente*, i `\newenvironment` lo fanno con il chiamante `\begin{nome}` ... `\end{nome}`. Vediamo.

1. Supponiamo che il nuovo ambiente sia
   `\newenvironment{xyw}[n.arg]{dich.apertura}{dich.chiusura}`
2. e che nel documento sia presente la chiamata
   `\begin{xyw}` *testo/oggetto* `\end{xyw}`
3. Consideriamo le specificità di `\newenvironment`[9]
   - Il `\newenvironment {xyw}` è equivalente alla coppia di comandi `\xyw`, `\endxyw` definiti dallo stesso `\newenvironment`. Alcuni autori li introducono nei loro pacchetti e classi in sostituzione del corrispondente `\newenvironment`.
   - I testi di sostituzione di `\xyw` e `\endxyw` sono le dichiarazioni di apertura e di chiusura del `\newenvironment`:
     ▷`\newcommand\xyw {dich.ap.}`,
     ▷`\def\endxyw{dich,ch.}`.
   - L'azione di `\begin{xyw}`*testo/oggetto*`\end{xyw}` inizia e chiude l'ambiente, chiama con `\begin{xyw}` il comando `\xyw` e con `\end{xyw}` il comando `\endxyw`.

Vediamo un esempio che conferma l'esistenza di quelle macro, dopo di che ci si potrà normalmente riferire ad esse.

[9] Vedi *The LaTeX 2ε Sources* p. 24, *The LaTeX Companion* p. 848, e-mail di Claudio Beccari.

1. Definiamo il nuovo ambiente
   ▷`\newenvironment{xyw}{\begin{quote}\itshape\small}{\end {quote}}` che teniamo non operativo: lo scriviamo in verbatim solo per averlo come riferimento.
2. Sono operative le definizioni dei comandi equivalenti al `\newenvironment`:
   ▷`\newcommand\xyw{\begin{quote}\small\itshape}`
   ▷`\def\endxyw{\end {quote}}`
3. Effettuiamo la chiamata `\begin{xyw}` alfa beta gamma delta epsilon eta theta iota kappa lambda mi ni xi phi omicron pi ro sigma tau chi psi omega. `\end{xyw}`, che non può che indirizzarsi alle due macro e che restitusce:

   alfa beta gamma delta epsilon eta theta iota kappa lambda mi ni xi phi omicron pi ro sigma tau chi psi omega.

Ma Lamport [10], che porta i suoi esempi di `\newenvironment` in termini di altri `environment` della base LaTeX, in riferimento a un `\newenvironment {xyw}{dich.ap.}{dich.ch.}`, ci dice che i processori TeX sostituiscono `\begin{xyw}` e `\end{xyw}` con le `dichiarazioni apertura` e di `chiusura` il che porterebbe a `\xyw` *testo/oggetto* `\endxyz`, che presa di per sé sembrerebbe molto simile alla coppia di macro alla TeX.

La tentazione era forte, ho preso i primi esempi di 4.1, ho sostituito i `\begin \end` con i comandi secondo le indicazioni qui sopra ed ecco il risultato istantaneo e lucente, funzionava! Sembrava proprio che tra le invocazioni di `\begin{xyw} \end{xyw}` e di `\xyw \endxyw` non ci fossero differenze e non ho minimamente pensato che i miei `\newenvironment` erano espressi in termini di `figure` e `table`. Proprio come chiedeva Lamport.

E ho mandato il file a Beccari per mostrargli che le cose potevano andare diversamente da come le pensava lui. Che ha risposto chiarendo al meglio la situazione, che riassumo.

Inizio risposta

1. Nella e-mail mi dice che i `\newenvironment` sono stati elaborati perché i `\begin{figure}` ... `\end{figure}` o i `\begin{table}` ... `\end{table}` presenti nel sorgente dei miei `\newenvironment` hanno provveduto a quanto avrebbero dovuto fare il `\begin{nome}` ... `\end{nome}`. L'elaborazione fatta dalla coppia di macro ha funzionato esclusivamente per quel motivo.
2. E allega un file che mostra con chiarezza la situazione:

   (a) Nel preambolo del file il codice del `\newenvironment`:
   ▷ `\newenvironment{prova}[1][.5\textwidth]{`

[10] Leslie Lamport a pag. 54 del suo *LaTeX User Guide and Reference*

```
\centering\small\sffamily\minipage{#1}
 {\endminipage\par}
```

(Dal codice di `prova` ho tolto la dichiarazione `\centering` che prevede di essere inserita in un gruppo/ambiente e che a seguito della seconda chiamata in cui l'ambiente non viene chiuso si espande in tutto il restante documento senza che sia riuscito a controllarla. Al contrario non è stato difficile fermare `\small` e `\sffamily`. Comunque, anche senza `\centering`, risulta chiaro quanto vuole evidenziare Beccari.)

(b) Nel documento scrive

```
\begin{prova}[.3\textwidth]
Questa è una minipage centrata che contiene un poco di testo, non molto, ma tanto da formare alcune righe.
\end{prova}

Questo è un poco di testo che segue l'ambiente: ha il rientro del capoverso, ma non è centrato, è composto con font con grazie ed è in corpo normale.

\prova[.3\textwidth] Questa è una minipage centrata che contiene un poco di testo, non molto, ma tanto da formare alcune righe. \endprova

E questo è un poco di testo che segue il non-ambiente, ma continua ad essere (centrato) in corpo small e composto con il carattere senza grazie.
```

(c) Lanciando le due chiamate, si ottiene:

Questa è una minipage centrata che contiene un poco di testo, non molto, ma tanto da formare alcune righe.

Questo è un poco di testo che segue l'ambiente: ha il rientro del capoverso, ma non è centrato, è composto con font con grazie ed è in corpo normale.

Questa è una minipage (centrata) che contiene un poco di testo, non molto, ma tanto da formare alcune righe.

E questo è un poco di testo che segue il non-ambiente, ma continua ad essere (centrato) in corpo small e composto con il carattere senza grazie.

Fine risposta

Sulla equivalenza Beccari aveva precedentemente accluso un suo commento.

Commento di Claudio Beccari

Bisogna osservare che il comando:

`\begin{xyw}` alfa beta gamma delta epsilon eta theta iota kappa lambda mi ni xi phi omicron pi ro sigma tau chi psi omega. `\end{xyw}`

non è equivalente a:

`\xyw` alfa beta gamma delta epsilon eta theta iota kappa lambda mi ni xi phi omicron pi ro sigma tau chi psi omega. `\endxyw`

Infatti `\begin{xyw}` non si limita ad eseguire il comando di apertura, ma prima apre un gruppo con `\begingroup`, poi carica anche una s.c. interna con il *nome dell'ambiente* `xyw` e, infine, esegue il comando `\xyw`[11]. Alla chiusura mediante il comando `\end{xyw}` viene dapprima controllato che l'ambiente da chiudere abbia lo stesso nome dell'ultimo ambiente che era stato aperto e il cui nome era stato memorizzato dentro alla s.c. interna, poi, se è così, viene eseguito il comando di chiusura `\endxyw` e infine viene chiuso il gruppo con `\endgroup` con il risultato di annullare tutte le assegnazioni e le definizioni eseguite dentro all'ambiente e di ripristinare le macro e i registri ai significati e contenuti che avevano prima dell'apertura di quell'ambiente; se il *nome dell'ambiente* non coincidesse con il nome dell'ultimo ambiente che era stato aperto, viene emesso un messaggio di errore che di solito è *fatale*, al quale non non si può porre rimedio se non terminando l'esecuzione del programma; i messaggi d'errore indicano il numero della riga dove si era aperto l'ultimo ambiente, quindi in generale non è difficile trovare la discrepanza fra i due nomi, sempre legata ad una distrazione dell'operatore, che spesso commette degli errori di battitura.

Beccari sottolinea anche che `\begin` attiva il comando di apertura attraverso un comando interno del nucleo di LaTeX, che in definitiva esegue indirettamente il comando `\xyz` attraverso `\csname xyz\endcsname`, e il comando `\end` esegue il comando di chiusura attraverso la stessa tecnica: `\csname endxyw\endcsname`. Il vantaggio consiste nel fatto che se l'ambiente `xyw` non è mai stato definito, passare attraverso la coppia di comandi `\csname` e `\endcsname` vuol dire formare un comando che ha una definizione predefinita pari a `\relax` a meno che il comando non abbia già una definizione per conto suo. Il comando `\relax` è innocuo, nel senso che non fa fare niente al programma di composizione. Grazie a questo "trucco" si possono trattare come ambienti tutte le dichiarazioni; per esempio è lecito scrivere `\begin{small}` testo `\end{small}`, che cambia il corpo del testo contenuto nell'ambiente e, anche se il comando `\endsmall` non è definito la costruzione `\csname endsmall\endcsname` manda in esecuzione il comando `\relax`; in sostanza viene controllato che l'ambiente da chiudere sia proprio `small`, viene eseguito un `\endsmall` fittizio, cioè il comando vero `\relax`, e viene chiuso il gruppo che era stato aperto con `\begin{small}`, col risultato di ripristinare automaticamente il corpo precedente. Certo è un modo un po' complicato di eseguire il comando `\small` confinato dentro a un gruppo ma, quando ci sono molti ambienti e gruppi annidati, la possibilità di errori è grande e la diagnostica offerta da `\begin` e `\end` torna molto utile.

Fine commento di Claudio Beccari

[11] Si noti la differenza fa il *nome dell'ambiente* `xyw` e il comando di apertura dell'ambiente `\xyw`.

Si può in qualche modo concludere affermando che se il `\newenvironment` predispone le istruzioni che agiranno sul *testo/oggetto* importato, la creazione vera e propria dell'ambiente e l'attivazione dei comandi di apertura e chiusura sono prerogativa del chiamante.

4.1 Esempi di `\newenvironment`[12]

Oltre agli aspetti generali e di gestione di cui è detto finora occorre sottolineare che il compito più specifico degli `environment` è l'elaborazione del *testo/oggetto* che il documento gli propone tramite il chiamante. È questo il motivo che porta a distinguere formalmente nel suo codice le dichiarazioni di apertura da quelle di chiusura. Normalmente il *testo/oggetto* viene a trovarsi tra le dichiarazioni di apertura e chiusura e questo basta per l'elaborazione sequenziale delle dichiarazioni.

Nei prossimi esempi il *testo/oggetto* viene sottoposto a più di una elaborazione e il primo passo sarà la memorizzazione del *testo/oggetto* in un registro box il che comporta che i gruppi espliciti delle dichiarazioni debbano essere violati per creare la porta d'ingresso sul registro (il primo registro box è l'ultimo elemento delle dichiarazioni di apertura). In presenza di registri TEX la porta viene creata da un gruppo implicito (come visto nelle le coppie TEX). Ma LATEX rende disponibile l'ambiente `lrbox`, uno strumento particolarmente utile che insieme alla definizione di uno specifico registro box individua e delimita (come un gruppo implicito) l'oggetto della registrazione (come il *testo/oggetto*). Il primo registro, sottoposto ad alcune elaborazioni, può essere memorizzato in un altro registro e solo dopo che sono terminate le operazioni sui registri si procede sequenzialmente sul restante delle dichiarazioni.

Gli esempi sono commentati rivolgendosi più alla gestione del codice che all'applicazione in sé, si tratta infatti di variazioni sul tema dell'incorniciatura di un testo, una tabella, un grafico del documento.

I primi due esempi elaborano con due registri box un *testo/tabella* introdotto dal documento. Il primo è gestito da registri e gruppi TEX annidati; il secondo da registri LATEX non annidati. Il terzo esempio elabora un *grafico* introdotto da un `\includegraphics` del documento con un solo registro box LATEX.

L'ultimo `\newenvironment` il cui *testo/tabella/grafico* del documento è vuoto ha la stessa struttura elaborativa degli altri casi e se da un lato è gestito dalla chiamata `\begin{}` *oggetto/testo/tabella* `\end{}` e quindi è un vero e proprio `environment`, dall'altro importando un *testo/tabella/grafico* vuoto è diverso. Infatti importa file esterni con un `\includegraphics` del suo codice, proprio come potrebbero fare una macro o un `\newcommand`.

FramedFig Per seguire lo schema di come il tutto a livello logico può funzionare consideriamo i comandi `\FramedFig`, `\endFramedFig` (equivalenti al `\newenvironment{FramedFig}`, vedi *Source2e*) e il chiamante `\begin{FramedFig}`*testo/oggetto*`\end{FramedFig}`.

[12] Non sarebbero stati scritti senza le e-mail di Claudio Beccari

`\begin {FramedFig}` chiama il comando `\FramedFig` che in tal modo, precedendolo, si troverà ad agire sul *testo/oggetto* mentre `\end{FramedFig}` chiama il comando `\endFramedFig` per portare a termine le istruzioni. E si può pensare che, all'interno dell'ambiente creato da `\begin{FramedFig}`... `\end{FramedFig}`, le cose stanno così: `\FramedFig` *testo/oggetto* `\endFramedFig`. Il primo testo di sostituzione a precedere il *testo/oggetto* e il secondo a seguirlo formano una lista di token che espansa esegue tutte le istruzioni. Per questo, in particolare con i registri, occorre che un'azione iniziata tra le dichiarazioni di apertura non si esaurisca in questo stesso gruppo. Per collocare correttamente il *testo/oggetto*, in questo caso una minipage, occorre che l'azione iniziata tra le dichiarazioni di apertura entri in quelle di chiusura terminandola nella posizione prevista. Qui, in presenza di registri TEX, il superamento dei confini delle dichiarazioni avviene grazie a due gruppi impliciti annidati.

I sorgenti del `\newenvironment{FramedFig}` e quello dei comandi equivalenti, vedi 4.

```
\newenvironment{FramedFig}[1]%
{\begin{figure}[thb]%
\lista={#1}%
\centering\setlength{\fboxsep}{0pt}\setlength{\fboxrule}{6pt}%
\setbox256\hbox\bgroup\setbox\figura\hbox\bgroup%
\color@setgroup}
{\color@endgroup\egroup\textcolor{blue}{\fbox{\relax%
\textcolor{yellow}{\fboxsep=4pt\fbox{\usebox{\figura}}}}}\egroup
\captionsetup{width=\wd256}\leavevmode\box256\relax%
\caption{\sffamily\the\lista}%
\end{figure}}
```

```
\newcommand\FramedFig[1]{%
\begin{figure}[thb]%
\lista={#1}%
\centering\setlength{\fboxsep}{0pt}\setlength{\fboxrule}{6pt}%
\setbox256\hbox\bgroup\setbox\figura\hbox\bgroup%
\color@setgroup}

\def\endFramedFig{\color@endgroup\egroup\textcolor{blue}{\fbox{\relax%
\textcolor{yellow}{\fboxsep=4pt\fbox{\usebox{\figura}}}}}\egroup%
\captionsetup{width=\wd256}\leavevmode\box256\relax
\caption{\sffamily\the\lista}%
\end{figure}}
```

Ora cerchiamo di guardare al codice.
Nelle dichiarazioni di apertura di `FramedFig` sono definiti due registri box TEX annidati, controllati dal comando primitivo `\hbox` compatibile con il gruppo implicito che iniziato con `\bgroup` tra le dichiarazioni di apertura e indipendentemente dal gruppo esplicito `{ }` può terminare con `\egroup` tra le dichiarazioni di chiusura .

Il codice delle dichiarazioni è articolato in varie istruzioni tra cui fondamentali quelle dei registri box annidati . Considerare per intero il testo

di sostituzione è complesso e di non facile gestione, è preferibile guardare, soprattutto a livello logico, alle azioni sui due registri una alla volta.

Considerando solo il testo di sostituzione relativo alla memorizzazione del *testo/oggetto*, che in questo caso è una minipage di larghezza assegnata contenente un testo, nel registro `\figura` si può scrivere[13]

```
{...\setbox\figura\hbox\bgroup\color@setgroup} \begin{minipage}
{7cm} { testo} \end{minipage} {\color@endgroup \egroup ...  }
```

Avvenuta la memorizzazione, il codice utilizzato viene rimosso dalla lista semplificando quello relativo al secondo registro che consiste nella incorniciatura dei dati del registro `\figura` il cui risultato va nella memoria del registro box 256. Le istruzioni essenziali per la strutturazione del registro box 256 e del suo gruppo implicito ora sono

```
{ ...\setbox256\hbox\bgroup}{ \textcolor{blue}{\fbox{\relax
\textcolor{yellow}{\fbox\fboxsep=4pt\usebox{\figura}}}}\egroup
...}
```

è il registro 256 a memorizzare i dati forniti dal gruppo implicito del suo `\hbox`: prima il colore della filettatura blu, poi di quella gialla, la distanza di separazione tra la cornice interna e la figura e anche i dati che il registro `\figura` gli passa con `\usebox`. Il gruppo delle dichiarazioni di chiusura è ancora aperto: con `\wd256` comunica a `\captionsetup` la larghezza della didascalia, esce con `\leavevmode` dalla modalità verticale creata dalla lista del registro 256 il cui contenuto con il comando `\box256` [14] costruisce l'output. Rimane da dire qualcosa sulla lista token. La didascalia della figura è memorizzata da TEX nel registro `\lista`, un registro `\toks`, che ne riceve il testo sotto forma di una lista di caratteri/simboli/sequenze di controllo ciascuno accompagnato dal proprio codice di categoria (token: coppia ordinata *byte,categoria*) che potrà essere ricomposta in righe secondo la dimensione orizzontale indicata in `\captionsetup`. Tra le dichiarazioni di apertura sono presenti anche le definizioni dello spessore delle cornici (`\setlength{\fboxrule}{6pt}`) e della loro distanza di separazione (`\setlength{\fboxsep}{0pt}`). `\color@setgroup}` e `\color@endgroup}` sono comandi del pacchetto `color` per un appropriato salvataggio dei colori in un registro.

Per invocare `FramedFig`:

```
\begin{FramedFig}{{\bfseries\small FramedFig}Primo articolo della costi-
 tuzione}
\begin{minipage}{7cm}
```

[13] Si stanno esaminando testi di sostituzione delle dichiarazioni di apertura e chiusura di cui non fanno parte le graffe dei gruppi delle dichiarazioni. Ciò nonostante sono presenti in rosso a ricordare uno dei motivi per la presenza delle graffe implicite e quello principale per la non semplicissima stesura del `\newenvironment`.

[14] Il flusso dei dati in uscita sembrerebbe dover avvenire nell'ordine inverso a quello di entrata, prima la figura poi la cornice. è l'accesso ai registri dimensionali del registro 256 che permette di seguire il flusso di lista nello stesso ordine in cui è stato costruito.

Figura 1: **FramedFig** Primo articolo della costituzione. Minipage importata.

```
{\large\bfseries L'Italia è una repubblica basata sul lavoro di pochi.}
\end{minipage}
\end{FramedFig}
```

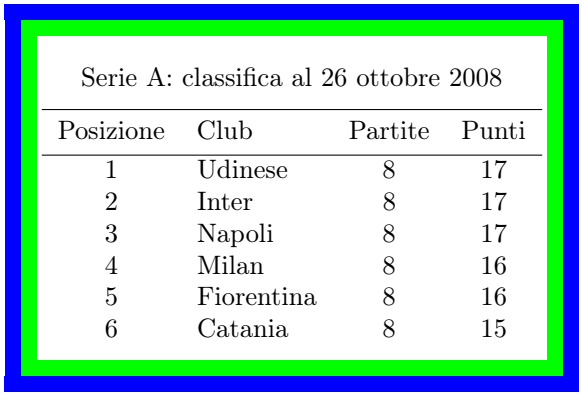
FramedFigLtx Il superamento dei limiti delle dichiarazioni di apertura e chiusura di un `\newenvironment` che usa registri box è possibile, oltre che con i gruppi impliciti `\bgroup` e `\egroup` collegabili soltanto ai comandi primitivi, anche con l'ambiente `lrbox` di LaTeX. Visto poi che i registri tendono a favorire la libertà del codice, in questo esempio saranno usati separatamente. Le dichiarazioni di apertura oltre alle inizializzazioni preliminari si limitano a memorizzare con `lrbox` nel registro `\figura` la tabella inclusa nell'ambiente `\begin{FramedFigLtx}` `\end{FramedFigLtx}` del documento.

Nel testo di sostituzione del gruppo di chiusura viene definito il registro LaTeX `\sbox{\oggetto}{ }` che memorizza la cornice blu e poi quella verde, che immette una separazione di 2pt tra tabella e cornice e che recupera con `\usebox{\figura}` i dati della tabella salvando il tutto nel registro `\oggetto`. Chiuso il registro, il primo `\usebox{\oggetto}` visualizza la figura incorniciata. Il secondo, utilizzato come argomento di `\settowidth{\largh}{\usebox\oggetto}`, ne estrae la larghezza che viene passata con `\captionsetup{width=\largh}` alla didascalia.

I sorgenti del `\newenvironment` `FramedFigLtx` e quello dei comandi equivalenti `(Tabella 1)`, vedi 4:

```
\newenvironment{FramedFigLtx}[1]%
 {\begin{table}[thb]%
  \lista={#1}%
  \centering\setlength{\fboxsep}{0pt}\setlength{\fboxrule}{6pt}%
   \begin{lrbox}{\figura}}
 {\end{lrbox}\sbox{\oggetto}{\textcolor{blue}{\fbox{\relax%
  \textcolor{green}{\fboxsep=2pt\fbox{\usebox{\figura}}}}}}%
  \usebox{\oggetto}%
  \settowidth{\largh}{\usebox\oggetto}%
  \captionsetup{width=\largh}\relax%
  \caption{\sffamily\the\lista}%
\end{table}}
```

```
\newcommand\FramedFigLtx[1]{%
  \begin{table}[bht]%
   \lista={#1}%
```



Serie A: classifica al 26 ottobre 2008

Posizione	Club	Partite	Punti
1	Udinese	8	17
2	Inter	8	17
3	Napoli	8	17
4	Milan	8	16
5	Fiorentina	8	16
6	Catania	8	15

Tabella 1: **FramedFigLtx** Con `\fboxsep=2pt` alla settima riga c'è una piccola separazione tra tabella e cornice e le righe orizzontali non toccano la cornice interna.

```
   \centering\setlength{\fboxsep}{0pt}\setlength{\fboxrule}{6pt}%
   \begin{lrbox}{\figura}}

\def\endFramedFigLtx{\end{lrbox}\sbox{\oggetto}{\textcolor{blue}{\fbox{\relax%
   \textcolor{green}{\fboxsep=2pt\fbox{\usebox{\figura}}}}}}% sep oggetto-cornice
   \usebox{\oggetto}%
   \settowidth{\largh}{\usebox{\oggetto}}%
   \captionsetup{width=\largh}\relax%
   \caption{\sffamily\the\lista}%
   \end{table}}
```

Per ottenere la `Tabella 1`:

```
\begin{FramedFigLtx}{{\bfseries\small  FramedFigLtx} Con \texttt{\string\fboxsep=2pt}
 alla settima riga  c'è una piccola separazione tra tabella e cornice  e
 le righe orizzontali non toccano la cornice interna.}
 \begin{tabular}{clcc}
 \multicolumn{4}{c}{ \rule [-3mm]{0mm}{9mm}Serie A: classifica al 26 ottobre
  2008
 \rule{0pt}{11pt}}\\
 \hline
   Posizione & Club &Partite & \rule{0pt}{11pt}Punti\\[0.5ex]
 \hline
   1 & Udinese &8 & 17\\
   2 & Inter &8 &17\\
   3 & Napoli&8&17\\
   4 & Milan&8&16\\
   5 & Fiorentina&8&16\\
   6 & Catania&8&15\\[3pt]
 \end{tabular}
\end{FramedFigLtx}
```

MinipInBox Un solo registro box, `\figura`, per memorizzare la minipage che ospiterà il grafico. La `minipage`, nel codice del `\newenvironment`, con-

trolla la larghezza della figura, i parametri migliorano la flessibilità del codice.

Il chiamante `\begin{MinipInBox} ... \end{MinipInBox}` latore anche di quattro argomenti, che in via preliminare vengono ordinatamente assegnati ai rispettivi parametri, è portatore del comando `\includegraphics` che di conseguenza viene a trovarsi tra il testo di sostituzione delle dichiarazioni di apertura e quello di chiusura. Il codice relativo al registro `\figura` (il parametro non è stato sostituito):

```
\begin{lrbox}{\figura}\begin{minipage}{#1} \includegraphics[width=
\textwidth]{figVoss5} \end{minipage}\end{lrbox}
```

Il primo passo è il controllo della larghezza del box e lo si fa definendo come ultima istruzione delle dichiarazioni di apertura `\begin{minipage}{#1}` la cui ampiezza orizzontale (lo `\hsize` del `\vbox`), assegnata col `#1`, controllerà la giustezza di quanto è immesso dal documento. In questo caso la minipage e quindi il registro `\figura` ospiterà il grafico importato da `\includegraphics` dove il valore attuale di `\textwidth`, essendo `\includegraphics` nell'ambiente della `minipage`, è `#1`.

Nelle dichiarazioni apertura si conferma sia l'uso, `\lista{#2}`, del registro `\toks` per la didascalia sia che le due cornici si tocchino, `\setlength{\fboxsep}{0pt}`, e che abbiano lo stesso spessore riservandosi di assegnarne il valore con `\setlength{\fboxrule}{#3}`.

C'è poi da assegnare la distanza di separazione fra la seconda cornice, la più interna, e la `minipage`/grafico tenendo presente che lo `\fboxsep` di pertinenza si trova nelle dichiarazioni di chiusura e non può ricevere assegnazioni dirette. Per farlo indirettamente si parte definendo nel preambolo una nuova lunghezza `\newlength{\dist}` e assegnando, tra le dichiarazioni di apertura, un valore a tale nuovo parametro (che funge da registro) con `\setlength{\dist}{#4}`. Ora `\fboxsep=\dist` posto tra le chiusure rende operativa l'assegnazione di questa distanza.

Rimane da sistemare la giustezza della didascalia e lo si fa nelle aperture con `\captionsetup{width=#1+#3+#3+#3+#3+2\dist}` non senza aver prima reso possibile la lettura di tale aritmetica richiedendo la disponibilità del pacchetto `calc` nel preambolo.[15]

Certo, con i due registri è tutto più tranquillo. Si potrebbe fare a meno di `\figura`? Noi sappiamo che per tracciare una cornice occorre avere un box le cui misure sono disponibili in un registro. Ma nel registro le ha messe TEX e allora?

Il sorgente di `MinipInBox (Figura 2)`:

```
\newenvironment{MinipInBox}[4]{%
 \begin{figure}[t]%
 \lista={#2}%
 \setlength{\dist}{#4}
```

[15] Assegnando `\fboxrule` prima di `\captionsetup` si può moltiplicare:`4\fboxrule`.

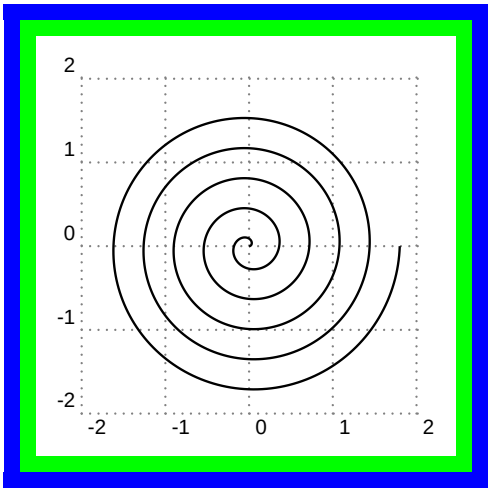


Figura 2: **MinipInBox** Immagine in una minipage di ampiezza assegnata e incorniciata, la minipage è nel sorgente. Ampiezza cornice 182.26378pt

```
  \captionsetup{width=#1+#3+#3+#3+#3+2\dist}\relax%
  \setlength{\ampz}{#1+#3+#3+#3+#3+#4+#4}
  \centering\setlength{\fboxsep}{0pt}\setlength{\fboxrule}{#3}%
    \begin{lrbox}{\figura}\begin{minipage}{#1}}
  {\end{minipage}\end{lrbox}{\textcolor{blue}{\fbox{\relax%
   \textcolor{green}{\fboxsep=\dist\fbox{\usebox{\figura}}}}}}
   \caption{\sffamily\the\lista \the\ampz}%
\end{figure}}
```

Per ottenere la `Figura 2`:

```
\begin{MinipInBox}{5cm}{{\bfseries\small MinipInBox} Immagine in una minipage di
   ampiezza assegnata e incorniciata, la minipage è nel sorgente.}{6pt}{8pt}
   \includegraphics[width=\textwidth]{figVoss5}
  \end{MinipInBox
```

FramedFigMcr Gli esempi precedenti si sono caratterizzati per la gestione del *testo/oggetto* del documento compreso tra `\begin{ambiente}` e `\end{ambiente}`. Le istruzioni della definizione di un `\newenvironment` agiscono però indipendentemente dalla presenza di un *testo/oggetto* e nel caso attuale dove il *testo/oggetto* è vuoto si ha un `\newenvironment` che, pur funzionando, di fatto si riduce ad un `\newcommand`. Sembra poi evidente che in questi `\newenvironment` cade la necessità di bypassare i limiti delle dichiarazioni di apertura e di chiusura e di usare `lrbox` o `\bgroup` e `\egroup`. Questa situazione si presta bene all'importazione di un grafico con `\includegraphics` posto direttamente tra le dichiarazioni di apertura.

Le operazioni che concernono `\sbox` e il suo registro `\figura`, oltre a quelle del registro token `\lista`, sono tutte fra le dichiarazioni di apertura separate da quelle del `\box256` che si trovano tra quelle di chiusura ma che, grazie ai registri, sono permeabili al flusso dei dati.

Alcune considerazioni sul codice.

Ha tre argomenti di cui uno facoltativo, il `#1`. L'argomento della variabile `\Larg` può non essere esplicitamente assegnato, e in tal caso varrà la larghezza del grafico importato da `\includegraphics`. Diversamente l'assegnazione della specifica larghezza è fatta da parte del codice che invoca il nuovo ambiente ponendo il valore scelto all'interno di una coppia di parentesi quadre. Il `#2` si riferisce al testo della didascalia che viene memorizzata nel registro token `\lista`. Il `#3` al nome del file del grafico da inserire. I tre argomenti vanno assegnati tra le definizioni di apertura.

Tra le definizioni di apertura, oltre a quelle dello spessore delle cornici (`\fboxrule`) e al loro spazio di separazione (`\fboxsep`), viene creato il registro box orizzontale `\figura` in cui sarà memorizzata la figura importata da `\includegraphics` prelevando, con la struttura condizionale `\ifx` di TEX a due opzioni, la dimensione originale oppure quella immessa come eventuale argomento opzionale, caricando il tutto in `\figura` la cui chiusura termina di fatto le definizioni di apertura.

Le operazioni di chiusura iniziano con la definizione del registro hbox a cui fanno capo le successive operazioni: che contornano il contenuto del registro `\figura` con una doppia cornice, definiscono i due colori, dispongono lo spessore delle cornici e la spaziatura grafico cornice, e salvano nel registro `\box256` la figura incorniciata. Il parametro `\wd256` nel frattempo registra la larghezza complessiva del suo box e con `\captionsetup` la fa diventare la larghezza della didascalia garantendo che sia la stessa della figura finale. A questo punto per ottenere l'output del registro `256` occorre lasciare il modo verticale. Lo si fa con `\leavevmode` e si invoca il `\box256` per copiare nel flusso di uscita il contenuto del suo registro e poi vuotarlo.

Il codice di `FramedFigMcr \texttt{(Figura 3)}`:

```
\newenvironment{FramedFigMcr}[3][]{\def\Larg{#1}%
 \begin{figure}[!t]%
  \lista={#2}%
  \centering\setlength{\fboxsep}{0pt}\setlength{\fboxrule}{6pt}%
  \sbox\figura{%
    \ifx\Larg\empty\includegraphics{#3}%    \else\includegraphics[width=#1]{#3}\fi}}
  {\setbox256\hbox{\textcolor{blue}{\fbox{\fboxsep=4pt\relax
   \textcolor{green}{\fbox{\usebox{\figura}}}}}}}
   \captionsetup{width=\wd256}\ampz=\wd256\leavevmode\box256\relax
   \caption{\sffamily\the\lista  \the\ampz}%
   \end{figure}}
```

Per ottenere la `Figura 3`:

```
 \begin{FramedFigMcr}[50mm]{{\bfseries\small FramedFigMcr}
```

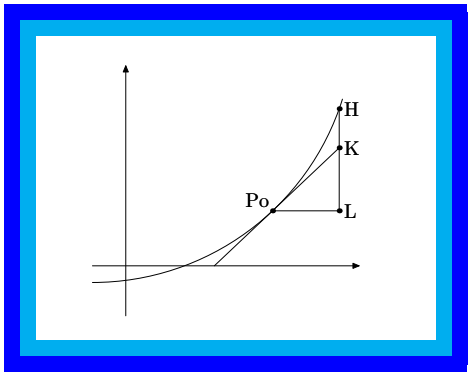


Figura 3: **FramedFigMcr** Didascalia con registro token; ampiezza della cornice 174.26378pt

```
        Didascalia  con registro \texttt{token};
        ampiezza  cornice.}{fig2b}
\end{FramedFigMcr}
```

4.2 Comandi e `\newcommand`

Un comando che si propone come `\&` stampa il simbolo `&` cui TEX attribuisce un significato speciale oppure sotto forma di sequenza di controllo, come una macro, indica un'azione su un determinato argomento che porterà alla stampa di un testo. Simboli come `# $ & %` sono essi stessi comandi. La base LATEX rende disponibile un ampio ventaglio di comandi pronti agli usi più disparati come `\include{filename}`, `\typeout{messaggio}`, `\setlength {textheight}{20cm}`, `\fbox{testo}`, `\texttt{testo}`. Ma ha anche un particolare tipo di comando, la `declaration`, che non crea un proprio testo ma agisce, ad esempio, cambiandone le dimensioni o lo stile sul testo di un ambiente o incluso in graffe. Dichiarazioni di uso frequente sono `{\small testo}`, `{\centering testo}`, `{\bfseries testo}`, `{\sffamily testo}`.

L'utente che al di là del panorama pressoché illimitato offerto da LATEX e dai suoi pacchetti vuole scriversi un comando per un compito specifico, oltre alle macro e ai `\newenvironment`, ha il `\newcommand{\nome}` con il vantaggio, rispetto alla flessibilità della macro, del controllo che il `\nome` non sia già stato usato e dell'univocità delle regole d'uso. Se poi si vuole saperne di più sia per gli aspetti di base che per quelli per pochi suggerisco di farsi guidare dall'indice analitico degli *Appunti ...* di Enrico Gregorio. Difficilmente si può fare meglio.

- Come primo esempio prendo la macro del punto 2.4.2 sostituendo a `\def` `\newcommand` e per la non riutilizzabilità dei nomi `\LetterHead` e `\Signature` con le loro traduzioni. Direi che il `\def\Signature##1{ ... }` della prima versione con il suo doppio segno di parametro è più significativo.

```
\newcommand\Intestazione[3]{%
 \raggedright{\textsf{#1}}\\
 \raggedright{\textsf{#2}}\\
 \raggedright{\textsf{#3}}\par
 $\vdots$
 \par
 \newcommand\Firma[1]{%
   \raggedright{Cordiali saluti}\\
   \raggedright{##1 \textit{#1}}\\
     }}

\Intestazione{Pinco Pallino}{viale dei Sogni 15}{55878 Chissà Dove}
\Firma{Il mago}
```

Pinco Pallino
viale dei Sogni 15
55878 Chissà Dove

⋮

Cordiali saluti
Il mago *Pinco Pallino*

4.2.1 Esempi di `\newcommand`

In questa sezione il `\newcommand` ritorna su due precedenti esempi risolti con il `\newenvironment`. Il `\newcommand{\CFramedFigMcr}` sembra più idoneo a gestire figure importate con `\includegraphics` di quanto facesse, almeno sul piano formale, il quasi omonimo ambiente. Il registro box `256`, `\setbox256`, salva la figura importata, il `\textcolor` più interno le associa un colore. I cinque argomenti permettono di gestire diverse grandezze. L'argomento `#1` è quello facoltativo associato all'ampiezza della figura gli altri sono `#2` nome del file.pdf da importare, `#3` primo colore, `#4` secondo colore, `#5` spessore cornice e spaziatura tra cornice e contenuto. Lo spazio tra le due cornici è zero. In questo esempio l'ampiezza della figura è assegnata, il testo della didascalia viene gestito direttamente dal documento con il comando `\caption`.

Se il `\newcommand` incornicia un testo non strutturato più lungo di una riga deve prima inserirlo inserirlo in un box verticale come il `\parbox{larghezza}{testo}` di LaTeX, alternativo ad una `minipage` o a un `\vbox`. è quanto viene fatto nell'esempio `\CommandImportesto[3]`: `#1` testo didascalia, `#2` testo importato, `#3` ampiezza orizzontale del `\parbox`. La necessità di creare un box verticale nel sorgente per accogliere un testo esiste anche nel `\newenvironment` dove però immettere un testo tramite ambiente sembra più naturale che trattarlo come argomento, specie se il testo è articolato.

1. Il sorgente del comando `\CFramedFigMcr`

```
\newcommand\CFramedFigMcr[5][]{%
\def\Larg{#1}%
\setbox256\hbox{\setlength{\fboxsep}{0pt}%
  \setlength{\fboxrule}{#5pt}%
  \textcolor{#3}{\fbox{\fboxsep=#5pt\textcolor{#4}%
  {\fbox{\textcolor{black}{%
    \ifx\Larg\empty
       \includegraphics{#2}%
    \else
    \includegraphics[width=#1]{#2}%
    \fi
     }}}}}%
  \captionsetup{width=\wd256}\leavevmode\box256\relax}
```

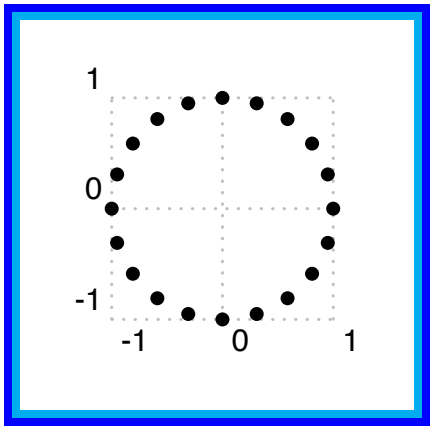


Figura 4: \CFramedFigMcr. Punteggiata circolare di raggio unitario importata con \includegraphics. Didascalia immessa dal documento.

Per ottenere la Figura 4:

```
\begin{figure}[h]
\centering
\CFramedFigMcr[50mm]{fig1}{blue}{cyan}{4}
\caption{{\bfseries\small
\symbol{92}CFramedFigMcr Punteggiata circolare di raggio unitario.
Didascalia immessa dal documento.}
\end{figure}
```

2. Il sorgente del comando \CommandImportesto

```
\newcommand\CommandImportesto[3]{
\begin{figure}[h]
 \lista{#1}
 \centering\setlength{\fboxrule}{4pt}\setlength{\fboxsep}{0pt}
```

```
 \sbox\scatola{\textcolor{cyan}{\fbox{\textcolor{orange}{\fboxsep=6pt%
 \fbox{\textcolor{blue}{\parbox{#3cm}{#2}}}}}}}
 \usebox\scatola
 \settowidth{\largh}{\usebox\scatola}
 \captionsetup{width=\largh}\relax%
 \caption{\sffamily\the\lista}
\end{figure}
}
```

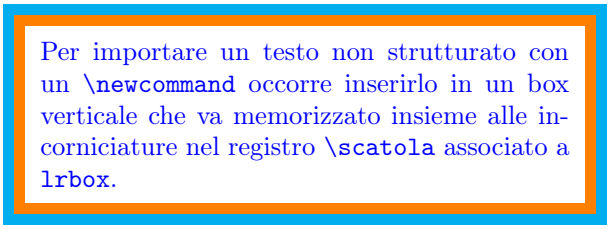


Figura 5: \CommandImportesto Il testo della didascalia passato al codice sorgente come argomento viene memorizzato nel registro \lista e distribuito da \caption sull'ampiezza della figura incorniciata.

Per ottenere la Figura 5:

```
\CommandImportesto{{\bfseries\small \symbol{92}CommandImportesto} Il testo
della didascalia passato al sorgente come argomento viene memorizzato nel
registro \texttt{\symbol{92}lista} e distribuito da \texttt{\symbol{92}caption}
sull'ampiezza della figura incorniciata.}{Per importare un testo non strutturato
con un \texttt{\symbol{92}newcommand} occorre inserirlo in un box verticale
che  va memorizzato insieme alle incorniciature nel registro \texttt{\symbol{92}
scatola} associato a \Progr{lrbox}.}{7}
```

5 Token in un registro

I token, cat 11 e 12, immessi in un registro box vengono memorizzati insieme allo spazio glue e i kern con l'obiettivo di essere stampati. I registri \toks memorizzano token di ogni categoria, non hanno strutture né dimensioni, e le sue eventuali sequenze di controllo non sono espandibili in questo ambiente.

Vediamo alcuni casi di token memorizzati nei registri \toks. Il comando \the\toks0 mostra il contenuto del registro dando il via all'espansione dei token della lista, che verranno anche visualizzati. Il contenuto del registro non è modificato dall'azione del comando \the.

1. Nel caso di \toks0={abc}, \toks0 contiene la lista abc, per cui \the\toks0= abc. Volendo aggiungere una stringa di caratteri in coda a quelli esistenti non basta assegnare \toks0={\toks0 mnp} in quanto le s.c. non espandono se assegnate ad un registro \toks e non si espande neanche nella forma di \the\toks0: cioè \toks0={\the\toks0 mnp} memorizza i token senza

espanderli. Infatti se abbiamo `\toks0={\the\toks0 mnp}` e andiamo a vedere qual è il contenuto del registro con il comando `\showthe\toks0`, nel file *console*, a mostrare la non avvenuta espansione, appare la lista `\the\toks0 mnp`.
Vediamo cosa accade con `\expandafter`.
▷ `\toks0={\the\toks0 mnp}`
▷ `\toks0=\expandafter {\the\toks0 mnp}`
Quando c'è un'assegnazione per un registro toks, TEX si aspetta una graffa aperta dopo di che i token vengono registrati senza essere espansi, fino ad incontrare la graffa chiusa. Ma TEX è flessibile e sa rimescolare le carte, con `\expandafter`. Con `\toks0=\expandafter{\the\toks0 mnp}` TEX non sa di essere di fronte ad una lista da ingerire e fa lavorare `\expandafter`, che è fuori dalla lista, salta la graffa ed attiva `\the` che espande il contenuto del registro e memorizza in `\toks0` la lista `abcmnp`.

2. Vediamo di fare la stessa cosa, aggiungere una stringa in coda ad una lista, servendoci delle macro.
▷ `\def\b{bravo}`
▷ `\edef\a{\b bravissimo}`
`\def\b{bravo}` ha il testo di sostituzione che per TEX è una lista di token. A tale lista se ne può aggiungere un'altra in coda con `\edef`. Infatti, con `\edef\a{\b bravissimo}`, `\edef` espande la macro `\b` del suo testo di sostituzione e, avendo `\edef` in tal modo concluso il suo compito, la macro sarà `\def\a{bravo bravissimo}` che, invocata, restituisce `\a`=bravo bravissimo.

3. Un caso con `\toks` e `\edef`.
▷`\def\xyz{mnp}`
▷`\toks5={\xyz\ abc}`
▷`\edef\lista{\the\toks5}`
`\showthe\toks5` mostra nei file *console* e *log* che il registro `\toks5` memorizza la lista dei token `\xyz`, `\ `, `a`, `b`, `c` . (I comandi `\show` e `\showthe` nel mostrare il testo di sostituzione di una macro e il contenuto del registro interrompono il processo di espansione che riprende con il tasto `return`. `\meaning` fa il lavoro di `\show` direttamente nel documento, vedi sotto Claudio Beccari.).

   Ma andiamo a vedere che `\edef` ha la caratteristica di agire su `\the\toks5` con un solo passo di espansione. Posto `\toks5={\xyz\ abc}` e `\edef\lista {\the\toks5}` si nota che `\edef` estrae i token dal registro senza espanderli ulteriormente ottenendo `\edef\lista {\the\toks5}` ⇒ `\def\lista {\xyz\ abc}` che non si espande ulteriormente. Controllando con `\show \lista` come stanno le cose per la macro `\lista`, `console` mostrerà `\show\lista = \xyz \ abc`. Si avrà un'ulteriore espansione se le s.c. saranno usate come chiamate.
   Invocando `\lista`, la macro si espande, e anche la sua s.c. `\xyz` viene rimpiazzata dal testo di sostituzione e si ha `\lista =  mnp abc`. A questo

punto neanche la lista dei token ha più la s.c.. infatti `\the\toks5 =` mnp abc.

Commento di Claudio Beccari

O meglio, sembra che `\toks5` non contenga più la macro se si vuole guardare il suo contenuto con il comando `\the`. In realtà la differenza fra quello che succede con `\edef` e quello che succede con `\the` è quanto segue:

- quando `\the` svolge il suo compito nel mostrare il contenuto del registro mette quanto estratto nel flusso delle cose da comporre; il motore di composizione si ritrova a dovere comporre i token estratti che formano la stringa `\xyz\ abc`; ma il token `\xyz` è un token complesso così come lo è il token `\␣`; perciò li esegue espandendoli e arriva alla sequenza di token non più espandibili `mnp␣abc` che fanno apparire come se il registro token non contenesse più la macro `\xyx` e la macro dello spazio.
- `\edef`, invece, esegue l'espansione del testo sostitutivo, ma la esegue solo in parte, nel senso che sostituisce ogni macro espandibile con la sua definizione, ma non prosegue oltre ad espandere le macro che potrebbero essere contenute nel testo sostitutivo. si confronti il codice seguente:

```
\toks5={\xyz\ abc}
\edef\lista{\the\toks5}
\verb|\tokz5| vale ``\the\toks5'' --\qquad--
\verb|\lista| È una \meaning\lista
```

Si ottiene:

`\tokz5` vale mnp abc – `\lista` È una macro:->\xyz \ abc

Come si vede, il comando `\edef` ha eseguito un solo passo di espansione del suo testo di sostituzione.

Fine commento di Claudio Beccari

4. Risolvendo il caso precedente con `\expandafter`. Con `\toks3=\expandafter {\xyz\ abc}`, TEX salta la graffa ed espande `\xyz`; sicché ora ingerisce una lista in cui la s.c. è rimpiazzata dal testo di sostituzione, e sia `\showthe\toks3` (in `console`) sia `\the\toks3` (nel documento) restituiscono mnp abc.

   Equivalenze:

```
\toks2={bravo}
\toks3={␣bravissimo}
\toks4=\expandafter{\the\toks2 \the\toks3}
```

   `\the\toks4 =` bravo␣bravissimo

```
\edef\tmp{\the\toks2 \the\toks3}
\toks4=\expandafter{\tmp}
```

   `\the\toks4 =` bravo␣bravissimo

5. Nel punto 1. si è visto come aggiungere una stringa in coda alla lista. Se poi si vuole mettere in testa alla lista il token `\xyz` occorrono due `\expandafter`: quello fuori dal gruppo esplicito porta il processore di input ad attivare lo `\expandafter` interno il quale a sua volta salta `\xyz` ed espande il comando `\the` che esplicita i token del registro. Ora l'assegnazione ha la forma `\toks6=\expandafter{\xyzabc}` del caso precedente ed `\expandafter` agisce sul comando `\xyz` che espandendosi viene rimpiazzato dal suo testo di sostituzione.

   ```
   \toks6={abc}
   \toks6=\expandafter{\expandafter \xyz\the\toks6}
   \the\toks6 =  mnpabc
   ```

6. Tra lista di token e testo di sostituzione di una macro senza parametri non c'è una grande differenza tanto da poterli anche usare uno per l'altro. A pag. 114 degli *Appunti* per aggiungere in coda a un testo di sostituzione di una macro altro testo si utilizzano i registri `\toks`. Si inizia definendo la macro `\abc` la cui s.c. viene inserita nella lista dei token per essere memorizzata nel registro `\toks4`, dopo essere stata espansa da `\expandafter`. Per stampare nel documento la lista aggiornata sotto forma di testo di sostituzione si ridefinisce la macro `\abc` con `\edef` assegnandogli per testo di sostituzione `\the\toks4`, che `\edef` espande nella lista di token del registro `\toks4`. Trattandosi di testo di sostituzione `\edef` è necessaria; ma avendo a che fare con una lista memorizzata nel registro si può visualizzarla direttamente con `\the\toks4`; o, limitandosi ad un controllo, con `\showthe\toks4` nel file *console*.

   ```
   \def\abc{abcd efgh}
   \toks4=\expandafter{\abc ijkl}
   \edef\abc{\the\toks4}
   \abc             = abcd efghijkl
   \the\toks4       = abcd efghijkl
   \showthe\toks4 -> abcd efghijkl
   ```

   Gli *Appunti* presentano una macro LaTeX che, assegnata la s.c. della macro contenente la lista iniziale e la stringa da aggiungere in coda, risolve direttamente la cosa. La macro iniziale va nuovamente assegnata.

   ```
   \def\abc{abcd efgh}
   \makeatletter
   \g@addto@macro\abc{ijkl}
   \makeatother
   ```

   `\abc` $\Rightarrow$ abcd efghijkl

   La definizione della macro, negli *Appunti*, non è complicata e ripercorre quanto abbiamo appena visto. Deve essere accompagnata dalla macro di riferimento, per noi `\abc`; il primo argomento è la s.c. della macro, il secondo è il testo da aggiungere.

```
\newcommand\g@addto@macro[2]{%
 \begingroup
   \toks@\expandafter{#1#2}%
     \xdef#1{\the\toks@}%
    \endgroup}
```

Gli *Appunti* presentano anche il `\newcommand \toksappend` che aggiunge alla fine di una lista contenuta in un registro token altri token. Il primo argomento è il nome del registro `\toks` il secondo è la lista dei token da aggiungere.

```
\newcommand\toksappend[2]{#1=\expandafter{\the#1#2}}
```

7. Macro per gestire liste di token sono a pag. 153 di *Tex by Topic*. Una è `\Prepend` che sembra *aggiungere* token in testa alla lista esistente, ma che in effetti *ricostruisce* la lista ponendo i nuovi elementi in testa. Inoltre per capire come funziona la macro occorre tener presente che se in un registro contenente una lista di token vengono immessi elementi di un'altra lista, questi sostituiscono quelli preesistenti. E poi se si sommassero non sarebbero necessari gli accorgimenti che stiamo esaminando. Si tratta di una macro con due parametri: il primo delimitato da `(to:)` il cui argomento è la lista de token da aggiiungere, il secondo parametro, non delimitato, riceverà il registro `\list`. La prima istruzione, `\toks0={#1}`, immette i nuovi token nel registro `\toks0`, segue la macro `\edef\act` il cui testo di sostituzione `\noexpand#2= {\the\toks0 \the#2}` è l'assegnazione al registro `\list` della lista formata dai nuovi token e da quelli della lista iniziale, che riceve sotto forma di secondo argomento. Si osserva poi che il primo membro `\list` della uguaglianza di `\edef` ha l'espansione bloccata per via del comando `\noexpand` mentre al secondo membro vengono espansi il `\toks0` dei nuovi token e a seguire il registro `\list`.
   La s.c. `\act`, non completamente espansa, non viene rimpiazzata dal testo di sostituzione; il comando `\show\act` ⇒ `\list={\a \b \c}`. `\showthe \list`, sempre nel file *console*, restituisce la lista `\a \b \c`.

```
\show\act
```

```
\newtoks\list
\list={\c}
```

```
\def\Prepend#1(to:)#2{\toks0={#1}
\edef\act{\noexpand#2={\the\toks0 \the#2}}
\act}
```

La chiamata alla macro:
`\Prepend \a \b (to:)\list`

6 Alla periferia di LaTeX

Si considerano due argomenti che possono essere considerati marginali per LaTeX: l'interattività e l'uso del Terminale.

6.1 `\typeout`, `\typein`, un esempio.

LaTeX ha i comandi `\typeout` e `\typein` che interagiscono col `Terminale`. Inseriti in documento con una sua struttura tipografica ne permettono l'utilizzazione a più utenti che abbiano prerogative o caratteristiche comuni. Personalmente la sto usando da vari anni per inviare via fax dal computer al medico di base le richieste di farmaci, esami e visite specialistiche mie e dei miei familiari.

Per quanto riguarda i dati dello 'studio medico' disposti nell'ambiente `flushlft` lo stile del font è ottenuto con la dichiarazione `\scshape`, mentre il comando TeX `\obeylines` dà il fine riga ai dati dello 'studio medico' in sostituzione degli alternativi `\\`.

Il comando `\typeout{messaggio}` fa stampare sullo schermo dal terminale il <messaggio> inviato, `\typein[\abc]{messaggio}` fa apparire sullo schermo il <messaggio> e il processore si ferma in attesa che l'utente abbia digitato la sua risposta al <messaggio>, risposta che diventa il testo di sostituzione della macro `\abc`. A questo punto <return> riattiverà il processore. Lamport propone `\typein` come `\typein[cmd]{msg}` equivalente a `\typeout{msg}` e `\newcommand{cmd}{typed input}` dove `cmd` indica una s.c. sinonimo di macro.

Nell'esempio che segue viene usata la struttura condizionale della divisione in casi (numerici), `\ifcase`, e per compiere le scelte necessarie vengono definite le s.c. `\assistito` e `\numprescr` che saranno anche le s.c. usate dai `\typein`. La prima scelta viene compiuta in base alla posizione occupata dal nome dell'assistito, la seconda per il numero dei farmaci (controlli, visite specialistiche), la terza per specificare il numero delle confezioni e il nome del farmaco (del controllo ...). Per mettere a punto le richieste (al massimo quattro) occorre definire altrettante sequenze di controllo che sono `\richiestauno`, `\richiestadue`, ..., `\richiestaquattro`. Il testo di sostituzione di queste s.c. servirà per stampare nel documento di output le varie richieste.

I primi due `\ifcase` sono inseriti in un `\loop` che esercita un controllo parziale ma significativo sui dati numerici che vengono immessi. Infatti la parte conclusiva ha la forma `\else\repeat` che diviene attiva se il numero immesso non rientra nello spettro dei casi previsti. Lo zero ne rimane fuori. Il `\repeat` collocato nell'ambiente `\else` del condizionale `\ifcase` ha la doppia funzione di terminare il condizionale e di attivare un nuovo ciclo di `\loop`. Ma è interessante osservare che `\repeat` si presta bene alla duplicità grazie alla definizione `\let\repeat=\fi`. Per `\ifcase` TeX legge `\fi`, per il `\loop` che ne prevede la presenza nella definizione è necessario il comando `\repeat`. Mossa vincente a costo quasi zero[16].

[16] Non dimenticando le risorse rese disponibili da i vari `\noexpand` ed `\expandafter` viene da chiedersi cosa avrebbe potuto fare Donald E. Knuth a un tavolo da gioco o in un consiglio di ministri.

C'è anche una chiusura con la data e il nome del richiedente ed un eventuale post scriptum con altre comunicazioni per il medico. Le relative scelte sono gestite con `\ifthenelse`.

Il preambolo.

```
\documentclass[a4paper, 12pt]{article}
\pagestyle{empty}
\usepackage{ifthen}
\usepackage{color}
\usepackage{soul}
\usepackage[latin1]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[italian]{babel}
\usepackage{textcomp,lmodern}
\usepackage{marvosym}
\frenchspacing
\newcount\numprescr
\newcount\assistito
\parindent=0pt
%==========================

\begin{document}


\begin{flushleft}
\obeylines
\scshape Per il
 Dott. EtaBeta
 Chissadove
 \FAX\  03256781
\end{flushleft}

 \vskip .7cm

% I comandi \typeout stampano nel terminale il testo racchiuso tra
% le graffe.
% I \typein[sequenza di controllo]{messaggio} memorizzano come testo
% di sostituzione del comando (inteso come sinonimo di macro)
% formalizzato dalla sequenza di controllo quanto sarà digitato
% nel terminale e che verrà utilizzato successivamente.
% \typein[\abc]{messaggio} equivale a \typeout{messaggio}
% \abc{testo digitato nel terminale}.
%
% In questo esempio per attivare  i comandi previsti occorre digitare
% -il numero corrispondente all'assistito,
% -il numero delle prescrizioni (non superiore a 4),
% -la quantità, nome e caratteristiche di ogni farmaco
% -e infine le eventuali note esplicative.
%
% Sono definiti come nuovi comandi \assistito, \numprescr,
```

```
% \richistauno/due/tre/quattro.
%
% Si utilizza il comando \ifcase controllato dal valore numerico associato
% alle s.c. \assistito o \numprescr e secondo il quale viene selezionato
% il caso. \else \repeat fa ripetere il loop nei casi numerici non previsti dai
% vari \or.
% La presenza di \repeat chiude in qualche modo il condizionale
% \ifcase, infatti lo \fi  che in genere chiude \ifcase in questo caso
% sarebbe segnalato errore.
% I due loop riducono e non eliminano (non coprono lo zero) le
% possibilità  di segnalazione di errore da parte  del compilatore.


%--- Digita nel terminale il numero corrispondente al richiedente ---
%--- e seleziona i suoi dati ----------------------------------------
\typeout{========================================}
%\assistito=-1
\loop
\typeout{Scegli il numero dell'assistito:}
\typeout{[1] Primo}
\typeout{[2] Seconda}
\typeout{[3] Terzo}

\typein[\assistito]{}

\ifcase \assistito
  %case 0
\or
\begin{minipage}[t]{7cm}
Assistito:   \hl{Primo Paziente} \par
    CF: ASSPRM61C11R007T
\end{minipage}
\or
\begin{minipage}[t]{7cm}
Assistito:   \hl{Seconda Impaziente} \par
    CF: ASSSCN71H61R007R
\end{minipage}
\or
\begin{minipage}[t]{7cm}
Assistito:   \hl{Nuovo Paziente} \par
CF: ASSNUV92D21R007A
\end{minipage}
\else
\repeat

\medskip

%---------------RICHIESTA FARMACI--------------
% Dal terminale digitare: prima il numero delle prescrizioni
% (da 1 a 4) (dopo digitato, il numero delle prescrizioni
```

```
% viene passato alla s.c. \numprescr, poi precisare  una
% alla volta i farmaci richiesti. Queste descrizioni divengono il
% testo di sostituzione di \richiestauno/due/tre/quattro.
%
% Si utilizza il comando \ifcase controllato dal valore numerico associato
% alla s.c. \numprescr e secondo il quale viene selezionato il caso.

\def\numprescr{}
\def\richiestauno{}
\def\richiestadue{}
\def\richiestatre{}
\def\richiestaquattro{}

 \loop
   \typein[\numprescr]{Digita il numero delle richieste, max 4:}

 \ifcase \numprescr
% case 0
  \or
\typein[\richiestauno]{PRIMA RICHIESTA}
  \or
\typein[\richiestauno]{PRIMA RICHIESTA}
    \typein[\richiestadue]{SECONDA RICHIESTA}
  \or
\typein[\richiestauno]{PRIMA RICHIESTA}
    \typein[\richiestadue]{SECONDA RICHIESTA}
    \typein[\richiestatre]{TERZA RICHIESTA}
  \or
\typein[\richiestauno]{PRIMA RICHIESTA}
    \typein[\richiestadue]{SECONDA RICHIESTA}
    \typein[\richiestatre]{TERZA RICHIESTA}
    \typein[\richiestaquattro]{QUARTA RICHIESTA}
    \else
    \repeat
%-------STAMPA L'ELENCO DELLE PRESCRIZIONI------
\vspace{4cm}

\noindent Richiesta per:
\ifcase \numprescr
%case 0
 \or
 \begin{enumerate}
   \item \richiestauno
  \end{enumerate}
\or
\begin{enumerate}
   \item \richiestauno
   \item \richiestadue
 \end{enumerate}
\or
```

```
\begin{enumerate}
   \item \richiestauno
   \item \richiestadue
   \item \richiestatre
 \end{enumerate}
 \or
\begin{enumerate}
   \item \richiestauno
   \item \richiestadue
   \item \richiestatre
   \item \richiestaquattro
 \end{enumerate}
\fi

% -----------------CHIUSURA-------------------
\vskip .5cm
\hfill\begin{minipage}{2in}
{\raggedright Grazie, \\
\ifthenelse{\equal{\assistito}{1}}{Primo Assistito}{%
 \ifthenelse{\equal{\assistito}{2}}{Seconda Assistita}{Nuovo Assistito}}}
\end{minipage}

\vskip 1 cm
La Miacittà, \today
\vskip 2cm
%-------------------POSTSCRIPTUM-----------------
% In questo caso \typein non ha altro compito se non quello
% di far digitare il testo dal terminale invece che nel documento,
% e la sua stampa sostituirà il messaggio POSTSCRIPTUM.
\typein[\risposta]{Vuoi aggiungere una nota per il medico? (si/no)}
\ifthenelse{\equal{\risposta}{si}}%
{\begin{itemize}\sffamily\item[PS:]\typein{POSTSCRIPTUM}
       \end{itemize}}{}
```

6.2 Gestione dei grafici col Terminale

Prendendo lo spunto dai grafici che appaiono nei `\newenvironment` e `\newcommand` vorrei accennare brevemente, a margine del LaTeX, alla capacità del `Terminale` di intervenire sui loro file. Disegnare un grafico è un'azione i cui risultati sono memorizzati in un file che successivamente sarà importato nel documento in cui ci interessa rappresentarlo. Il pacchetto che si usa per disegnare può rappresentare il grafico in un'area adeguata a contenerlo e allora il passaggio nel documento finale può avvenire senza ulteriori interventi. Con alcuni pacchetti questo non avviene: si potrà ottenere un documento pdf ospitante solo il grafico oppure un documento in formato ps e allora nel primo caso occorre scontornare l'immagine e nel secondo cambiare il formato. Operazioni che possono essere

facilmente eseguite dal Terminale[17].

Con il Terminale si entra nell'ambiente di una versione Unix basata su BSD 4.4 il cui termine *directory* indica un file in cui si memorizzano informazioni su altri file, quello che per il Mac è una cartella . Lanciandolo, il Terminale indica la propria *home directory* e segnala di essere operativo operativo.

Nel mio caso la segnalazione è:

```
Macintosh:~ macmini$
```

e il simbolo ~ di dice che la *home* è la *directory* macmini che ci permette di accedere alle proprie subdirectory. E lanciando un comando dalla *home* macmini diventa la *directory* di lavoro. La finestra del Terminale è gestita da uno specifico programma denominato *shell*. Le *directory* del computer sono distribuite e collegate secondo lo schema di un albero rovesciato che parte dalla *root*, con vari rami e livelli.

Con il Terminale che indica `Macintosh:~ macmini$` il comando `pwd` (print working directory) riporta il percorso (pathname) assoluto della directory di lavoro:

```
Macintosh:~ macmini$ pwd
/Users/macmini
```

dove la prima / rappresenta la radice, Users è la seconda *directory* (rispetto alla radice) e la *directory* macmini è l'ultima di un percorso assoluto che inizia dalla *directory* radice.

Vediamo i comandi `date` e `ls`:

Il comando `date` ha il significato che ci si aspetta.

```
Macintosh:~ macmini$ date
Gio 28 Lug 2011 15:05:32 CEST
Macintosh:~ macmini$
```

Il comando `ls` elenca i file e le *directory* presenti nella *home* a cui ho aggiunto una copia della *directory* TeX così da poter raggiungere i file grafici con un percorso relativo che parte da macmini:

```
Last login: Fri Aug 26 13:42:24 on ttys000
Macintosh:~ macmini$ ls
Context Manual  Dropbox    LINEAR   Photos     TeX
Desktop         Faxes      Library  Pictures   riccardo
Documents       ILife      Movies   Public
Downloads       Italcogim  Music    Sites
Macintosh:~ macmini$
```

I file dei grafici in genere si trovano in una specifica *directory*, diversa dalla *home* iniziale, e che dovrà diventare la nuova *directory* di lavoro. Operazione per la quale è disponibile il comando `cd` (change directory) con il quale si dovrà costruire un percorso che raggiunge la cartella contenente i file da elaborare. Se si inizia dalla directory di lavoro (finora la *home*) si avrà un percorso relativo.

[17] Accennerò al Terminale del sistema operativo OS X del Mac, Il riferimento è *Il Terminale e Unix per Mac OS X* di Dave Taylor & Jerry Peek.

Poiché si vuole intervenire su file grafici della directory (cartella) immagini si dovrà digitare ogni passo del percorso *home* - /immagini (la prima *directory*, per noi TeX, a segnalare che si ratta di un percorso relativo deve essere scritta senza la barra rovescia). Il *pathname*, a partire dalla *home* sarà:

```
Macintosh:~ macmini$ cd TeX/arstexnicaUno/InserGraf/immagini
Macintosh:immagini macmini$
```

Il terminale con la scrittura `<Macintosh:immagini macmini$>` ci comunica che, nel computer con la directory macmini come *home*, la la directory di lavoro è la cartella immagini.
Per avere i file della *directory* immagini si lancia la linea di comando `Macintosh:immagini macmini$ ls` che ci restituisce l'elenco dei file della *directory* di lavoro:

2tabelle.aux ellpar.ps spira2.dvi
2tabelle.dvi ellpar.tex spira2.eps
2tabelle.eps ellpol.aux spira2.epsi
2tabelle.epsi ellpol.dvi spira2.log
2tabelle.log ellpol.eps spira2.pdf
2tabelle.pdf ellpol.epsi spira2.ps
2tabelle.ps ellpol.log spira2.tex
2tabelle.tex ellpol.pdf spira3.aux
circon1.aux ellpol.ps spira3.dvi
circon1.dvi ellpol.tex spira3.eps
circon1.eps figVoss5.aux spira3.log
circon1.log figVoss5.dvi spira3.pdf
circon1.pdf figVoss5.epsi spira3.ps
circon1.ps figVoss5.log spira3.tex
circon1.tex figVoss5.pdf spirale.aux
circon2.aux figVoss5.ps spirale.dvi
circon2.dvi figVoss5.tex spirale.eps
circon2.eps figVoss5copia.dvi spirale.log
circon2.epsi figVoss6.eps spirale.pdf
circon2.log figVoss6.pdf spirale.ps
circon2.pdf iperbole.aux spirale.tex
circon2.ps iperbole.dvi stringa.aux
circon2.tex iperbole.eps stringa.dvi
circon3.aux iperbole.epsi stringa.eps
circon3.dvi iperbole.log stringa.log
circon3.eps iperbole.pdf stringa.pdf
circon3.epsi iperbole.ps stringa.ps
circon3.log iperbole.tex stringa.tex
circon3.pdf spira1.aux tabellatex.aux
circon3.ps spira1.dvi tabellatex.dvi
circon3.tex spira1.eps tabellatex.eps
ellpar.aux spira1.epsi tabellatex.log

```
ellpar.dvi    spira1.log    tabellatex.out
ellpar.eps    spira1.pdf    tabellatex.pdf
ellpar.epsi   spira1.ps     tabellatex.ps
ellpar.log    spira1.tex    tabellatex.tex
ellpar.pdf    spira2.aux
```

Con il terminale che ci segnala `Macintosh:immagini macmini$`
per aprire un particolare file occorre un comando del tipo:
`Macintosh:immagini macmini$ open tabellatex.pdf`
E il Terminale ritorna in `Macintosh:immagini macmini$`.

Finora abbiamo visto percorsi relativi, a partire dalla *home directory*, ma il comando `pwd` ce ne mostra il percorso completo, il *pathname* assoluto, che inizia dalla radice dell'albero delle *directory*, la *root*, indicata dalla /.
`Macintosh:immagini macmini$ pwd` ci dà il percorso completo:
`/Users/macmini/TeX/arstexnicaUno/InserGraf/immagini`
E il Terminale ci segnala il proprio stato: `Macintosh:immagini macmini$`
Per riportare il Terminale alla *home* basterà:
`Macintosh:immagini macmini$ cd`
`Macintosh:~ macmini$`.

Torniamo alla cartella di lavoro immagini e vediamo una operazione specifica. Nella cartella immagini è presente il file `stringa.pdf` che consiste in una pagina pdf contenente soltanto l'immagine di una stringa, immagine che vogliamo scontornare preparandola ad essere importata in un altro documento. L'utility `pdfcrop` del pacchetto omonimo (sensibile alla filettatura `\color{white}` tracciata da `\fbox` e alla distanza di separazione di `\fboxsep` eventualmente assegnate nel documento originale) fa al caso nostro e nel terminale daremo il conseguente comando
`Macintosh:immagini macmini$ pdfcrop stringa`
che restituisce una immagine.pdf scontornata denominata 'stringa-crop.pdf', per distinguerla dall'originale che infatti è presente nella cartella `immagini`:

```
PDFCROP 1.23, 2010/01/09 - Copyright (c) 2002-2010 by Heiko Oberdiek.
 ==> 1 page written on 'stringa-crop.pdf'.
```

Nella stessa cartella è presente anche il file `spira2.ps` (formato `PostScript`) che si vuole portare in `pdf`. Si può utilizzare l'*utilty* `ps2eps` che riporta le coordinate di due vertici opposti del rettangolo contenitore, avendo l'avvertenza di inserire l'opzione -f (force) per sostituire il file omonimo se già presente, e avremo un file `eps` che `Anteprima` traduce direttamente in `pdf` (lo si può fare esplicitamente dal Terminale con il convertitore `ps2pdf`):

```
Macintosh:immagini macmini$ ps2eps -f spira2.ps
Input files: spira2.ps
Processing: spira2.ps
```

```
Rendering with existing ->BoundingBox: 0 0 612 792
Calculating Bounding Box...ready.->BoundingBox: 166 531 290 645
Creating output file spira2.eps...ready.
```

In effetti guardando a come lo OS X distribuisce le cartelle nel Macintosh si nota che la strada maestra per raggiungere la cartella delle immagini è quella che inizia dalla cartella delle Applicazioni (Applications) in cui, tra le altre, OS X immette la cartella tex. La *directory* Applications si trova, come seconda *directory* (seconda rispetto alla root), in un ramo parallelo alla *directory* di Users e quindi anche alla *directory* macmini. Con la linea di comando:
`Macintosh:~ macmini$ cd /applications/tex/arstexnicauno/insergraf/immagini`
⇒ `Macintosh:immagini macmini$`
che con il comando `Macintosh:immagini macmini$ pwd` ci conferma il percorso assoluto:
`/applications/tex/arstexnicauno/insergraf/immagini` (parallelo a quello visto sopra e che esclude macmini).
Il comando `Macintosh:immagini macmini$ ls` ci restituisce lo stesso elenco visto sopra con l'aggiunta del file 'stringa-crop.pdf'.

Per lasciare il Terminale è opportuno scollegarsi col comando `exit`
`Macintosh:immagini macmini$ exit`
Il Terminale risponderà: `logout [Processo completato]`

Indice analitico