Apuntes de LaTeX

Capítulo 8: Nociones de Programación LaTeX

El compilador TeX contiene aproximadamente 300 secuencias de control (comandos) llamadas *primitivas*. Éstas son operaciones de bajo nivel que no pueden ser descompuestas en acciones más simples. El resto de lo que propiamente se llama TeX, unas 600 instrucciones, son "macros", es decir, comandos definidos a partir de las 300 primitivas, haciendo uso de las capacidades de compilador (es decir, lenguaje de programación) de TeX. Asimismo, el procesador de textos LaTeX es otro conjunto de macros construidas a partir de comandos TeX. En éste capítulo se introducirán las herramientas básicas de programación disponibles en TeX, útiles para definir nuevos comandos ó entornos, modificar parámetros, automatizar tareas, en definitiva personalizar nuestro documento.

1. Nuevos Comandos y Entornos

1.1. Comandos

En ésta sección describiremos cómo utilizar el comando `\newcommand` para definir nuevos comandos LaTeX que puedan ayudarnos a simplificar el realizar tareas repetitivas. Repasaremos ahora el uso de este tipo de comandos, desde una perspectiva más formal. Para la definición de un nuevo comando se dispone de tres posibilidades:

```
\newcommand{\NombreComando}[NumArg][ArgDefecto]{Definición}
\renewcommand{\NombreComando}[NumArg][ArgDefecto]{Definición}
\providecommand{\NombreComando}[NumArg][ArgDefecto]{Definición}
```

donde `\NombreComando` es el nombre que queremos asignar al nuevo comando, `NumArg` indica el número de argumentos que va a tener (comprendido entre 1 y 9), `ArgDefecto` es el valor por defecto de un argumento optativo (el primero de ellos), y `Definición` contiene la definición del comando, donde los distintos argumentos se denotan como `#1, #2, etc...`

Entre estas tres versiones existen diferencias importantes. `\newcommand` se utiliza para definir *nuevos* comandos, por lo que debemos estar seguros de que el comando a definir no existe. `\renewcommand` se utiliza para *redefinir* comandos ya existentes, reescribiendo y borrado la definición anterior del comando. Finalmente, `\providecommand` define el nuevo comando *sólo en el caso de que el comando no exista*; en caso contrario la nueva definición carece de efecto.

Para cada una de estas tres posibilidades existen versiones con y sin asterisco; las versiones con asterisco (`\newcommand*{\NombreComando}[NumArg]{ArgDef}{Def}`, etc...)

no permiten que los argumentos puedan extenderse a más de un párrafo, mientras que las versiones sin asterisco (`\newcommand{\NombreComando}[NumArg]{ArgDef}{Def}`, etc...) permiten que los argumentos se extiendan a más de un párrafo.

Ejemplos:

- Imaginemos que queremos que un texto aparezca con tipo de letra sansserif e itálico; podemos entonces definir el comando `\nuevotipo`, dependiente de un parámetro (el texto a cambiar de tipo):
  `\newcommand{\nuevotipo}[1]{{\itshape\sffamily #1}}`
  tras lo cual, escribiendo `\nuevotipo{texto sansserif}` obtendríamos *texto sansserif*.

- Cambiemos ahora el ejemplo anterior; supongamos que, además, se quiere que, por defecto, el texto aparezca en tamaño `\large`, aunque ésto último sea también una opción modificable; definiríamos entonces:
  `\newcommand{\nuevotipo}[2][\large]{{#1\itshape\sffamily #2}}`
  lo cual hace que escribiendo `\nuevotipo{texto sansserif}` resulte *texto sansserif*, mientras que con `\nuevotipo[\small]{texto sansserif}` obtendríamos *texto sansserif*

- Veamos ahora otro ejemplo útil para la escritura de expresiones matemáticas; imaginemos que la expresión $(x_1, x_2, \ldots, x_n)$ aparece frecuentemente en nuestro documento. Podemos entonces definir:

  ```
  \newcommand{\vect}{(x_1,x_2,\dots,x_n)}
  ```

  con lo cual, cada vez que escribamos `$\vect$` (el nombre del nuevo comando) se imprimirá $(x_1, x_2, \ldots, x_n)$. Todos los nuevos comandos conviene situarlos *en el preámbulo*.

- Ahora compliquemos un poco el ejemplo con la introducción de *argumentos variables*. Si por ejemplo escribimos:

  ```
  \newcommand{\vect}[1]{(#1_1,#1_2,\dots,#1_n)}
  ```

  (añadiendo un argumento, que se sustituye en la fórmula con `"#1"`), escribiendo `$\vect{x}$` obtendríamos $(x_1, x_2, \ldots, x_n)$, con `$\vect{a}$` se tendría $(a_1, a_2, \ldots, a_n)$, etc...

- Añadiendo más argumentos, podemos obtener construcciones más complejas, por ejemplo, definiendo:

  ```
  \newcommand{\vect}[2]{(#1_1,#1_2,\dots,#1_#2)}
  ```

  `$\vect{x}{n}$` daría como resultado $(x_1, x_2, \ldots, x_n)$ mientras que con `$\vect{a}{p}$` se obtendría $(a_1, a_2, \ldots, a_p)$.

- Practiquemos ahora la definición de comandos con argumentos optativos, que toman un determinado valor por defecto. Por ejemplo, construyamos:

  ```
  \newcommand{\nuevovector}[2][x]{(#1_1,#1_2,\dots,#1_#2)}
  ```

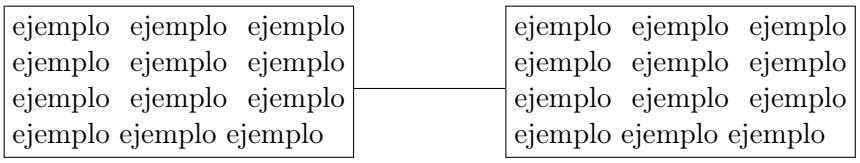
  donde la "x" entre paréntesis es el valor por defecto del argumento opcional (siempre

el número 1). Entonces, escribiendo: `$\nuevovector{n}$` ó `$\nuevovector{p}$` obtendríamos $(x_1, x_2, \ldots, x_n)$ y $(x_1, x_2, \ldots, x_p)$ respectivamente, mientras que añadiendo un argumento optativo cambiaríamos su valor por defecto de "x": `$\nuevovector[a]{n}$` $\longrightarrow (a_1, a_2, \ldots, a_n)$.

Ejercicio: Escribir un comando dependiente de dos argumentos que calcule la derivada parcial respecto de una función f respecto a una variable x, de forma que se puedan elegir tanto f como x:

$$\frac{\partial f}{\partial x}$$

Ejercicio: Definir un nuevo comando, dependiente de dos argumentos (párrafos de texto) que los coloque enmarcados, con anchura `0.3\textwidth` por defecto (cada párrafo), aunque modificable, unidos por una línea de longitud 2cm, y todo ello centrado;



1.2. Entornos

También es posible definir nuevos entornos, o redefinir entornos ya existentes; para ello se dispone de los siguientes comandos:

```
\newenvironment{NombreEntorno}[NumArg][ArgDef]{DefEntrada}{DefSalida}
\renewenvironment{NombreEntorno}[NumArg][ArgDef]{DefEntrada}{DefSalida}
```

que funcionan de un modo similar a los comandos del tipo `\newcommand`, en cuanto a que admiten argumentos (hasta 9), opcionalmente con el primero de ellos optativo. La diferencia reside en que en el argumento `DefEntrada` se indican las órdenes que se deben ejecutar *antes de entrar en el entorno*, y en el argumento `DefSalida` la que se deben ejecutar *al salir del entorno*. Una vez definido el nuevo entorno, se debe utilizar de la siguiente forma:

```
\begin{NuevoEntorno}{Arg1}...{ArgN}
Texto y comandos
\end{NuevoEntorno}
```

Al igual que en el caso de los comandos, existen versiones sin y con asterisco, con el mismo significado, es decir, que respectivamente admiten ó no argumentos de más de un párrafo.

Por ejemplo, construyamos un entorno que cree una minipágina de anchura variable (por defecto media página), centrada, y con el texto en negrita:

```
\newenvironment{mientorno}[1][0.5]{\begin{center}\begin{minipage}%
{#1\textwidth}\bfseries}{\end{minipage}\end{center}}
```

y tras definir éste nuevo entorno, tecleando:

```
\begin{mientorno}
Ejemplo de texto con una anchura estándar de media página, centrado,
y en tipo de letra negrita
\end{mientorno}
```

obtenemos:

Ejemplo de texto con una anchura estándar de media página, centrado, y en tipo de letra negrita

ó, si queremos emplear el argumento optativo y reducir la anchura del texto a 0.3 veces la anchura de texto (`\textwidth`):

```
\begin{mientorno}[0.3]
Ejemplo de texto con una anchura de un tercio de página, centrado, y
en tipo de letra negrita
\end{mientorno}
```

Ejemplo de texto con una anchura de un tercio de página, centrado, y en tipo de letra negrita

Es importante tener en cuenta que los argumentos de un entorno sólo pueden utilizarse en la definición de entrada (`DefEntrada`). Si los necesitamos en la definición de salida, podemos utilizar el "truco" de guardarlos convenientemente, empleando un comando `\newcommand` para ello. En el siguiente ejemplo, creamos un entorno `cita` para escribir citas, dando el nombre del autor como argumento:

```
\newenvironment{cita}[1]{\newcommand{\autor}{#1}%
\begin{quote}\itshape‘‘}{’’\end{quote}\centerline{\autor}}
```

Tras lo cual, por ejemplo:

```
\begin{cita}{Andres Fernández}
Nuestras vidas son los ríos que van a parar al mar, que es el morir
\end{cita}
```

produce:

“ Nuestras vidas son los ríos que van a parar al mar, que es el morir ”

Andres Fernández

2. Compilación por trozos: `\input` e `\include`

Imaginemos que estamos escribiendo un documento largo (un libro, por ejemplo). Es conveniente, a la hora de depurar errores, escribir y compilar cada parte por separado. Para ello LaTeX proporciona dos posibilidades:

- El comando `input{Fichero.tex}` produce que el compilador, al encontrar esta instrucción, lee el fichero indicado en el argumento y continúa compilando dicho fichero. En el argumento del comando podemos dar, si el fichero no se encuentra en el directorio actual, el camino hasta él. Debe tenerse cuidado de que instrucciones clave como `\documentclass` ó `\begin{document}` no se dupliquen. Entonces, para escribir un libro, por ejemplo, podemos tener un documento con la siguiente estructura:

```
\documentclass[opciones]{book}
\usepackage{paquete1}
.....
\begin{document}
%\input{capitulo1.tex}
%\input{capitulo2.tex}
%\input{capitulo3.tex}
......
\end{document}
```

  y, a la hora de depurar errores, descomentar individualmente cada una de las líneas `\input{fichero.tex}`. También es posible utilizar este comando para otros usos, por ejemplo, incluir listas de instrucciones `\newcommand` y personalizaciones diversas que podamos querer hacer comunes a varios documentos.

- Una alternativa más cómoda es utilizar, en vez de `\input`, el comando `\include{Fichero}` (es esencial omitir la extensión .tex en éste caso). Entonces, en el preámbulo se puede colocar el comando `\includeonly{Fichero1,Fichero2,...}`, que hace que sólo se incluyan en la compilación los ficheros que aparecen en el argumento. Es importante mencionar que al comenzar y terminar, la orden `\include` induce un salto de página (más exactamente, un comando `\clearpage`, que además "expulsa" elementos flotantes pendientes), por lo que esta alternativa es conveniente utilizarla sólo para incluir capítulos de un libro ó tesis.

3. Conceptos básicos sobre contadores y longitudes

3.1. Contadores

En su funcionamiento habitual, LaTeX utiliza un amplio número de contadores con el fin de enumerar distintos elementos de un documento: páginas, secciones, tablas, figuras, etc... Cada contador tiene un *nombre* que permite identificarlo; así, `page` es el contador que identifica páginas, `chapter` capítulos, etc... En lo sucesivo, denotaremos ese nombre como *NombreContador*. Cada contador lleva asociados una serie de elementos de diferente significado: *nombre*, *valor* (siempre un número entero) y *formato*, éste último pudiendo tomar variadas formas: (I, II, III..., a, b, c...)

Se dispone de los siguientes formatos de contador:

`\arabic{NombreContador}`	1, 2, 3, 4...
`\alph{NombreContador}`	a, b, c, d... (nota 1)
`\Alph{NombreContador}`	A, B, C, D... (nota 1)
`\roman{NombreContador}`	I, II, III, IV... (nota 2)
`\Roman{NombreContador}`	I, II, III, IV...
`\fnsymbol{NombreContador}`	*, **, ***, ****... (nota 3)

Nota 1: El valor del contador no puede superar 27 (número de letras en el abecedario)

Nota 2: El resultado mostrado es el que se obtiene con babel, opción spanish. Sin ello, se obtendría i, ii, iii, ...

Nota 3: Igualmente, el resultado mostrado es el obtenido con babel y spanish; en caso contrario, se utilizan las marcas inglesas: *, †, ‡... En ambos casos, el valor no puede ser superior a 6

Asociado a cada contador existe un comando, llamado **representación del contador**, que permite imprimir el valor del contador *NombreContador* en alguno de los formatos descritos; el comando es:
`\the`*NombreContador*

Cuando LaTeX define un nuevo contador, le asigna inicialmente la representación correspondiente al formato `\arabic`; si queremos cambiarla, podemos redefinirla mediante el comando `\renewcommand*`; veamos unos ejemplos de lo que se puede hacer:

```
Este ejemplo muestra cómo obtener
el número de la página en curso;
ésta página es la número \thepage, en
la representación original.\\
\renewcommand*{\thepage}{\roman{page}}
Ahora esta cambiada a números romanos:
Página \thepage \\
Finalmente, algo más elaborado: \\
\renewcommand*{\thepage}{[Sección %
\thesection \ -- Página \arabic{page}]}
Estamos en: \thepage
```

Este ejemplo muestra cómo obtener el número de la página en curso; ésta página es la número 6, en la representación original.
Ahora esta cambiada a números romanos: Página VI
Finalmente, algo más elaborado:
Estamos en: [Sección 3 – Página 6]

Ejercicio: Cambiar la representación de las secciones en éste documento a números romanos (I, II, III...), y la de las subsecciones al formato: I-a, II-b, etc...

Podemos cambiar los valores de un contador con los siguientes comandos:

- `\setcounter{NombreContador}{Valor}` ⟶ Asigna al contador *NombreContador* el valor entero `Valor`, con independencia del valor anterior
- `\addtocounter{NombreContador}{Valor}` ⟶ Incrementa *NombreContador* con la cantidad `Valor`, que puede ser positiva ó negativa.

Ejemplo:

```
Esta es la sección \thesection. Pero
podemos añadirle 2 fácilmente:\\
\addtocounter{section}{2}
ahora estamos en la sección \thesection
Mejor lo dejamos como estaba, porque
si no (comprobar comentando la
línea siguiente) las restantes
secciones quedarían numeradas
incorrectamente (esto es, el efecto
de estos cambios de numeración
es \emph{global}):
\addtocounter{section}{-2}
```

Esta es la sección 3. Pero podemos añadirle 2 fácilmente:
ahora estamos en la sección 5. Mejor lo dejamos como estaba, porque si no (comprobar comentando la línea siguiente) las restantes secciones quedarían numeradas incorrectamente (esto es, el efecto de estos cambios de numeración es *global*):

Podemos recuperar el valor *numérico* de un contador, independientemente de su representación, con el comando:

```
\value{NombreContador}
```

lo cual es útil para la gestión e contadores, como veremos a continuación.

Se definen nuevos contadores con la instrucción:

```
\newcounter{NuevoContador}[ContadorExistente]
```

que introduce un contador de nombre `NuevoContador`, y le asigna cero como valor inicial. El argumento `ContadorExistente` es *optativo*, y sirve para subordinar `NuevoContador` al contador ya existente `ContadorExistente`, de la misma forma que, por ejemplo, el contador *subsection* está subordinado al contador *section*: incrementar en una unidad el contador *section* inplica que el contador *subsection* se reinicia a cero automáticamente.

Veamos un ejemplo de cómo introducir un nuevo contador, con el fin de enlazar varias listas `enumerate` manteniendo la numeración (`enumi` es el contador estándar LaTeX para los ítems de primer nivel en entornos enumerate):

```
Las primeras lecciones
son las siguientes:
\begin{enumerate}
 \item Números reales
 \item Números complejos
 \setcounter{conserva}{\value{enumi}}
\end{enumerate}
Mas adelante, se estudiarán
temas más complicados:
\begin{enumerate}
 \setcounter{enumi}{\value{conserva}}
 \item Continuidad
 \item Derivación
\end{enumerate}
```

Las primeras lecciones son las siguientes:

1. Números reales
2. Números complejos

Mas adelante, se estudiarán temas más complicados:

3. Continuidad
4. Derivación

Debe mencionarse que al crear un nuevo contador se crea automáticamente el comando `\the`*NuevoContador*, con la definición `\arabic`{*NuevoContador*} por defecto.

Cuando se modifica un contador con los comandos `\setcounter` y `\addtocounter`, los contadores subordinados no se ponen a cero; para obtener ése efecto, se dispone de los comandos:

`\stepcounter{NombreContador}` ⟶ Incrementa `NombreContador` en una unidad, y reinicia todos los contadores subordinados a él

`\refstepcounter{NombreContador}` ⟶ Lo mismo que el anterior, pero declarando también como valor del comando `\ref` el texto generado por `\the`*NuevoContador* cuando se utilizan referencias cruzadas con los comandos `\label` y `\ref`

Veamos un pequeño ejemplo que ilustra como utilizar estos comandos; definimos:

```
\newcounter{prg}[section]\newcounter{linea}[prg]
\newcommand*{\lin}{%
\addtocounter{linea}{1}\thelinea\quad}
\renewcommand*{\theprg}{\arabic{section}.\arabic{prg}}
\newenvironment*{programa}{%
\refstepcounter{prg}
\begin{center}Programa~\theprg\end{center}
\ttfamily\obeylines\obeyspaces}{\par}
```

Ahora, como ejercicio, escribir un pequeño documento `article`, con un par de secciones y un par de entornos `programa` en cada una de ellas, utilizando dentro del entorno `programa` el comando `\lin` para ir enumerando líneas de código de un programa. Asimismo, incluir una referencia cruzada a uno de los programas (Ver archivo ejercicio2.pdf adjunto)

3.2. Longitudes

Al igual que con los contadores, LaTeX es también capaz de crear y modificar variables de tipo Longitud. Las longitudes que habitualmente utiliza LaTeX pueden tomar dos tipos de valores:

Rígidos: Toman un valor determinado; por ejemplo:
`\quad` = 11.747 pt, `\thinspace` = 1.958 pt, `\hoffset` = -28.45274pt [1]

Elásticos: Toman un valor que LaTeX puede modificar dentro de unos límites, a fin de optimizar la composición del documento. Por ejemplo, `\bigskip`, `\medskip` y `\smallskip`

El *comando* `\bigskip` se define como:

```
\vspace{12pt plus 4pt minus 4pt}
```

lo cual quiere decir que LaTeX debe introducir un espacio vertical de 12 pt, aunque tiene la libertad de incrementarlo ó reducirlo en 4pt, según convenga a fin de distribuir el espacio de forma homogénea. Alguna de las holguras `plus` ó `minus` pueden estar ausentes en la definición, pero si ambas aparecen deben estar en ése orden.

[1] Hay que tener cuidado en no confundir el concepto de *longitud* y del *valor* que toma una longitud; `\hoffset` es una longitud, mientras que `\quad` y `\thinspace` son *comandos* que dejan en blanco un espacio horizontal de valor rígido

Los comandos `\bigskip`, `\medskip` y `\smallskip`, respectivamente, tienen asociadas longitudes elásticas con valores almacenados en `\bigskipamount`, `\medskipamount` y `\smallskipamount`, por los que tales comandos se definirían de hecho como:

`\bigskip` ⟶ `\vspace{\bigskipamount}`

`\medskip` ⟶ `\vspace{\medskipamount}`

`\smallskip` ⟶ `\vspace{\smallskipamount}`

y donde cada una de éstas longitudes elásticas toma valores:

`\bigskipamount` :: 12.0pt plus 4.0pt minus 4.0pt

`\medskipamount` :: 6.0pt plus 2.0pt minus 2.0pt

`\smallskipamount` :: 3.0pt plus 1.0pt minus 1.0pt

Puede obtenerse el valor de cualquier longitud con el comando:

`\the\NombreLongitud`

donde `NombreLongitud` es el nombre de la longitud; éste comando siempre expresa las longitudes en unidades pt, con el punto como separador decimal.

Al igual que ocurría con los contadores, los valores de una longitud pueden modificarse. Existen dos comandos para ello:

- `\setlength{\NombreLongitud}{Valor}` Asigna a la longitud `\NombreLongitud` un valor igual al argumento `Valor`, que debe ser una longitud (ésto es, expresada en unidades cm, pt, etc...). Puede ser un valor tanto rígido como elástico (por ejemplo, `5mm plus 1mm minus 2mm`). También es posible que `Valor` sea una variable de longitud (`\textwidth`) con quizás un factor multiplicativo (`0.5\textwidth`, por ejemplo).

  Una forma alternativa de asignar a `\NombreLongitud` un valor es utilizar la sintaxis:

  `\NombreLongitud=Valor` ó bien: `\NombreLongitud Valor`

- `\addtolength{\NombreLongitud}{Valor}` Suma a la longitud `\NombreLongitud` la cantidad `Valor`, que puede ser positiva ó negativa.

Al contrario que lo que ocurría con los contadores, cuyas asignaciones tienen carácter *global* (es decir, trascienden el grupo dentro del cual han sido declaradas, y tienen efecto en todo el resto del documento), las asignaciones de longitud tienen por defecto carácter *local*; si se realizan dentro de un grupo, el valor anterior a la asignación se recupera a la salida del grupo. En el caso de que deseemos un efecto global, puede ser aconsejable realizar tales asignaciones en el preámbulo del documento.

Se pueden definir nuevas longitudes con el comando:

`\newlength{\NuevaLongitud}`

que crea una nueva longitud llamada `\NuevaLongitud`; es importante que `\NuevaLongitud` no sea ni un comando ni una longitud LaTeX ya existentes, en cuyo caso obtendríamos un mensaje de error. Por defecto, las nuevas longitudes son creadas con un valor inicial 0.0 pt.

Para la gestión de valores de longitud son útiles los siguientes comandos:

`\settowidth{\NombreLongitud}{Objeto}`

`\settoheight{\NombreLongitud}{Objeto}`

`\settodepth{\NombreLongitud}{Objeto}`

que calculan, respectivamente, la anchura (width), altura (height) y profundidad (depth) de un objeto, asignando el valor resultante a la longitud `\NombreLongitud`.

Ejemplos:

Imaginemos que queremos medir la longitud asociada al comando `\quad`. Para ello podemos definir una nueva longitud:

`\newlength{\longi}`

a continuación, asociamos a `\longi` la anchura del espacio asociado al comando `\quad`:

`\settowidth{\longi}{\quad}`

tras lo cual, el comando `\the\longi` muestra el valor 10.88788pt

Ahora creamos otra longitud: `\newlength{\longitud}`

que empleamos para medir la anchura, altura, y profundidad de la palabra Pajarita

```
\noindent La anchura de la
palabra {\Large Pajarita}
es \settowidth{\longitud}%
{\Large Pajarita} \the\longitud\\
su altura es \settoheight{\longitud}%
{\Large Pajarita} \the\longitud\\
su profundidad \settodepth{\longitud}%
{\Large Pajarita} \the\longitud\\
```

La anchura de la palabra Pajarita
es 48.91916pt
su altura es 9.91672pt
su profundidad 2.80008pt

```
\newlength{\longA}
\settowidth{\longA}{xxxxx}
\begin{center}
xxxxx\\
xxxxx\hspace{\longA}xxxxx\\
xxxxx\hspace{\longA}xxxxx%
\hspace{\longA}xxxxx\\
xxxxx\hspace{\longA}xxxxx\\
xxxxx
\end{center}
```

xxxxx
xxxxx xxxxx
xxxxx xxxxx xxxxx
xxxxx xxxxx
xxxxx

3.2.1. Longitudes elásticas `fil`

En este apartado describiremos dos *unidades de longitud* elásticas:

- `fil`
- `fill`

que LaTeX utiliza para introducir espacios de longitud variable. Ambas proporcionan dos diferentes grados de "elasticidad infinita"; `fil` es una unidad de longitud elástica infinitamente más grande que cualquier longitud rígida, mientras que `fill` es infinitamente más grande que `fil` (y por tanto, que cualquier longitud rígida).

Basados en éstas unidades de longitud, existe una variedad de comandos:

- `\fill` Es una *longitud*, de valor `0pt plus 1fill`
- `\stretch{n}` Es una *longitud* de valor 0pt y holgura un número `n` de unidades `fill` (entero ó decimal). Así, `\fill` equivale a `\stretch{1}`

De éste modo, los comandos `\hfill` y `\vfill` equivalen a `\hspace{\fill}` y `\vspace{\fill}`, respectivamente. La utilidad del comando `\stretch{n}` está en la posibilidad de separar objetos con espacios proporcionales a diversas cantidades. Véase el siguiente ejemplo:

```
Colocamos un texto centrado:\\[2mm]
\vrule\hspace{\stretch{1}}Texto
centrado\hspace{\stretch{1}}\vrule\par
Ahora colocamos un texto con
el doble de espacio a
un lado que al otro:\par
\noindent\vrule\hspace{\stretch{1}}%
Texto\hspace{\stretch{2}}\vrule\par
Otro ejemplo, con la distancia
entre T1 y T2 igual a tres veces
la distancia a los márgenes:\par
\noindent\vrule\hspace{\stretch{1}}
T1\hspace{\stretch{3}}T2
\hspace{\stretch{1}}\vrule
```

Colocamos un texto centrado:

| Texto centrado |

Ahora colocamos un texto con el doble de espacio a un lado que al otro:

| Texto |

Otro ejemplo, con la distancia entre T1 y T2 igual a tres veces la distancia a los márgenes:

| T1 T2 |

(para imprimir la barra vertical de referencia al comienzo y final de línea en el ejemplo anterior, hemos utilizado el comando `\vrule`; podemos poner una "marca" en blanco con los comandos `\mbox{}` ó `\null`).

- `\hfill` y `\vfill` (ya descritos)

- `\hfil` y `\vfil` Análogos a los anteriores, pero empleando para la elasticidad una unidad `fil` en lugar de `fill`.

El siguiente ejemplo ilustra la diferencia entre las unidades `fil` y `fill`:

```
\noindent A \hfil B \hfil C \\
D \hfill E \hfill F \par
```

A B C
D E F

¿Porqué cambian las posiciones de B y C en la primera línea? La respuesta está en que, antes de cortar una línea, LaTeX introduce un espacio de elasticidad variable, a fin de evitar que las líneas cortas se estiren hacia la derecha. Éste espacio se controla a través de la longitud `\parfillskip`, que por defecto tiene el valor `0pt plus 1fil`. Por tanto, en el primer ejemplo se equilibran los espacios asociados a tres comandos `\hfil`. En el segundo caso, esto no sucede, dado que `\hfill` corresponde a un grado de elasticidad infinitamente más grande.

- `\hfilneg` y `\vfilneg` Equivalen, respectivamente, a `\hspace{0pt plus -1fil}` y a `\vspace{0pt plus -1fil}`, y permiten **cancelar** el efecto de los comandos `\hfil` y `\vfil`; por ejemplo:

```
\parindent=0pt \parfillskip=0pt
\newcommand*{\centrar}[1]{%
\vrule\hfil #1\hfil\vrule}
\centrar{Centrado}\par
\centrar{Centrado anulado\hfilneg}\par
\centrar{\hfilneg Centrado anulado}
```

|Centrado|
|Centrado anulado|
|Centrado anulado|

- `\hss` Equivale a `\hspace{0pt plus 1fil minus 1fil}`, e interviene en la definición de los comandos `\leftline`, `\rightline` y `\centerline`
- `\vss` Análogo vertical, que equivale a `\vspace{0pt plus 1fil minus 1fil}`

Los siguientes comandos (algunos de ellos ya mencionados anteriormente) tienen un efecto similar a `\hfill`, con la diferencia de que en el espacio intermedio introducen diversos símbolos de extensión variable (en dirección horizontal):

- `\hrulefill` ⟶ Raya

```
A\hrulefill B\hrulefill C
```

A_____B_____C

- `\dotfill` ⟶ Línea de puntos

```
A\dotfill B\dotfill C
```

A.....B.....C

- `\downbracefill` y `\upbracefill` ⟶ Llaves hacia abajo ó hacia arriba, respectivamente.
- `\leftarrowfill` y `\rightarrowfill` ⟶ Flechas a izquierda y derecha, respectivamente.

```
\parindent=0pt \parfillskip=0pt
\mbox{}\hspace{\stretch{1}}A%
\hspace{\stretch{3}}\mbox{}\\[-3pt]
\mbox{}\downbracefill\mbox{}%
\hspace{\stretch{2.3}}\mbox{}\\
B\hspace{\stretch{1}}C%
\hspace{\stretch{1}}D\\[-7pt]
\mbox{}\hspace{\stretch{2.3}}%
\mbox{}\upbracefill\mbox{}\\
\mbox{}\hspace{\stretch{1}}E%
\hspace{\stretch{2}}F%
\hspace{\stretch{1}}\mbox{}\\[-5pt]
\mbox{}\hspace{\stretch{0.5}}%
\rightarrowfill%
\hspace{\stretch{0.5}}\mbox{}\\
```

A
B C D
E F
⟶

Finalmente, describiremos los comandos `\rlap{Objeto}` y `\llap{Objeto}`; respectivamente, colocan `Objeto` en una caja de anchura 0pt (por lo que el cursor no se mueve), con el objeto saliendo hacia la derecha ó izquierda de la caja. Por ejemplo:

```
Tachamos la palabra izquierda%
\llap{\rule[2.5pt]{48pt}{0.4pt}}
y seguimos escribiendo.\par
\hfil \rlap{uno}\llap{dos}\vrule \par
\hfil \llap{dos}\rlap{uno}\vrule
```

Tachamos la palabra ~~izquierda~~ y seguimos escribiendo.

dosuno

dosuno

Ejercicio: Diseñar un comando, dependiente de un argumento, que tache un fragmento de texto.

~~pepe y juan~~

4. Programación con TEX

4.1. Otro modo de definir comandos

Anteriormente hemos visto cómo definir nuevos comandos mediante la utilización de los comandos tipo `\newcommand` de LATEX. Existe un modo alternativo, empleando

comandos de más bajo nivel de TEX (de hecho, éste es el modo estándar de contruir "macros" LATEX). Para ello existe el comando `\def`, con la siguiente sintaxis:

```
\def\NuevoComando#1...#9{Definición}
```

donde `\NuevoComando` es el nombre del nuevo comando, `#1...#9` los argumentos de que depende (hasta 9), y entre llaves su definición. Por ejemplo, definamos:

```
\def\ecuacion#1#2{\ensuremath{#1_1^2+#1_2^2+\cdots+#1_#2^2=1}}
```

tras lo cual, `\ecuacion{z}{5}` produce $z_1^2 + z_2^2 + \cdots + z_5^2 = 1$. Nótese el uso del comando `\ensuremath{Fórmula}`, que tiene como resultado asegurar que la expresión `Fórmula` se ejecuta dentro del modo matemático (por lo cual, no es necesario abrir y cerrar signos $ antes y después del comando `\ecuacion`).

El comando `\def`, a diferencia del `\newcommand`, permite elegir los delimitadores de los argumentos (que con `\newcommand` siempre deben ser llaves, ó corchetes para los argumentos optativos). Además, es incluso posible prescindir de las llaves al escribir los argumentos del comando; por ejemplo, en el caso anterior podríamos haber escrito `\ecuacion z5` con el mismo resultado que `\ecuacion{z}{5}`: TEX lee secuencialmente los argumentos tras el nombre del comando. Para elegir delimitadores especiales entre los argumentos, simplemente los incluimos entre `#1`, `#2`,.. `#n`. Por ejemplo, redefinimos `\ecuacion` como:

```
\def\ecuacion#1;#2:{\ensuremath{#1_1^2+#1_2^2+\cdots+#1_#2^2=1}}
```

lo cual indica que el primer argumento debe terminar con ";" y el segundo con ":"

ahora debemos escribir `\ecuacion z;5:` para obtener $z_1^2 + z_2^2 + \cdots + z_5^2 = 1$

4.2. Definiciones globales

Todos los comandos creados con `\newcommand`, `\providecommand` ó `\def`, (ó redefinidos con `\renewcommand`) son *locales*; es decir, si están definidos dentro de un grupo, su acción estará restringida a ése grupo. En el ejemplo siguiente se ve cómo la redefinición del comando `\prueba` dentro del entorno `itemize` carece de efecto fuera de él:

```
\def\prueba{Prueba 1}
\begin{itemize}
\def\prueba{Prueba 2}
\item \prueba
\end{itemize}
\prueba
```

- Prueba 2

Prueba 1

Si queremos definir un comando global (con efecto fuera del grupo donde es definido) se puede utilizar cualquiera de estas dos alternativas:

```
\global\def\NuevoCOmando#1...#9{Definición}
```

```
\gdef\NuevoComando#1...#9{Definición}
```

es decir, o bien anteponemos el comando `\global` a la definición, o bien usamos el comando `\gdef`. EL comando `\global` también puede usarse para hacer globales otro tipo de asignaciones de tipo local, como por ejemplo las modificaciones de longitudes (`\setlength` y `\addtolength`)

Ejercicio: Reemplazar en el ejemplo anterior la definición de `\prueba` dentro del entorno `itemize` por una de carácter global, y observar el cambio en el resultado.

4.3. Definiciones recursivas: el comando `\edef`

Cuando se define un comando, no se ejecutan los comandos que puedan participar en la definición. Por ejemplo, si tan sólo definimos `\def\a{\b}`, siendo `\b` un comando inexistente, la compilación no produce errores. Sólo cuando intentemos *utilizar* el nuevo comando `\a` obtendremos un error. En ocasiones, puede interesar que al realizar la definición de un nuevo comando, se ejecuten las instrucciones que participan en la definición. Imaginemos, por ejemplo, que queremos redefinir un comando `\a` de forma recursiva, es decir, de forma que la nueva definición de `\a` contenga al mismo `"\a"` en su definición:

```
\def\a{hola}  \def\a{\a \a \a}  \a
```

las instrucciones anteriores provocan un error (comprobar). La forma de resolver el problema es redefinir el comando `\a` con el comando `\edef`, totalmente análogo a `\def`, pero que hace que al redefinir `\a` se expanda ó ejecute antes el comando `\a` dentro de la definición:

```
\def\a{hola}  \edef\a{\a \a \a}  \a
```

produce ahora: holaholahola

Si queremos que el comando `\edef` tenga efecto global, se debe utilizar el comando `\xdef`, que equivale a `\global\edef`.

4.4. El comando `\let`

Imaginemos que definimos un comando en función de otros comandos, y los comandos en los que se basa cambian. Este cambio se trasladará entonces al nuevo comando:

```
\noindent\def\uno{1}Uno: \uno \\
\def\dos{\uno\uno} Dos: \dos \\
\def\uno{uno} Uno: \uno \ Dos: \dos
```

Uno: 1
Dos: 11
Uno: uno Dos: unouno

En ocasiones, puede necesitarse definir un comando que sea independiente de los cambios que se produzcan en los comandos sobre los que está definido. Con esta utilidad está construido el comando `\let`, que "saca una copia" de un comando para que funcione siempre de la misma manera, con independencia de redefiniciones posteriores de comandos. Se utiliza con la sintaxis:

```
\let\NuevoComando=\ComandoExistente
```

que puede usarse también en caso de comandos con argumentos (cuidando de que el comando antiguo y su "copia" tengan el mismo número de argumentos). El ejemplo siguiente ilustra el funcionamiento de `\let`:

```
\noindent\def\uno{1}Uno: \uno \\
\def\dos{\uno\uno} Dos: \dos \\
\let\UNO=\uno \def\DOS{\UNO\UNO}
\def\uno{uno} Uno: \uno \\
Dos: \dos \ \ DOS: \DOS
```

Uno: 1
Dos: 11
Uno: uno
Dos: unouno DOS: 11

4.5. Manipulación de contadores y longitudes a través de TEX

Veamos ahora cómo se trabaja con contadores y longitudes desde el punto de vista de TEX. Se pueden realizar operaciones con tres tipos de magnitudes:

Contadores Corresponden a registros tipo `count`, y se definen con el comando: `\newcount\NuevoContador`; el registro puede almacenar números enteros entre -214783647 y +214783647

Longitudes rígidas Corresponden a registros tipo `dimen`, y se definen con el comando: `\newdimen\NuevaLongitud`

Longitudes elásticas Existen dos tipos de registro:
- `skip`: se definen con `\newskip\NuevaLongitud`
- `muskip`: análogo de longitud elástica, que se utiliza sólo en el modo matemático; se definen con `\newmuskip\NuevaLongitud`

Existen comandos para realizar las cuatro operaciones aritméticas básicas (suma, resta, multiplicación y división) con todos los registros anteriores (ambas sintaxis, con `advance` ó `advance by`, etc..., son equivalentes):

- `\advance\NombreRegistro` ±`Número` `\advance\NombreRegistro by` ±`Número` donde `Número` debe ser una longitud, si tratamos con registros de tipo longitud, ó un número entero, si trabajamos con un contador.
- `\multiply\NombreRegistro` ±`Número` `\multiply\NombreRegistro by` ±`Número` donde `Número` debe ser *siempre* un entero.
- `\divide\NombreRegistro` ±`Número` `\divide\NombreRegistro by` ±`Número` `Número` también debe de ser un entero. En el caso de un contador, se almacenará la parte entera de la división; en el caso de longitudes, éstas se transforman primero a unidades sp (la más pequeña de TEX; 1 sp = 65536 pt) y el resultado se redondea a un múltiplo entero de ésta unidad.

Veamos unos ejemplos:

```
\newskip\LongElastica
\LongElastica=%
10pt plus 1fill minus 2fill%
\par \the\LongElastica
\advance\LongElastica by %
5pt plus 3fill minus 1fill%
\par \the\LongElastica
\multiply\LongElastica by 3%
\par \the\LongElastica
\divide\LongElastica by 2
\par \the\LongElastica
```

10.0pt plus 1.0fill minus 2.0fill
15.0pt plus 4.0fill minus 3.0fill
45.0pt plus 12.0fill minus 9.0fill
22.5pt plus 6.0fill minus 4.5fill

En el caso de longitudes *rígidas*, existe un forma alternativa de multiplicarlas por un factor: `\Longitud1=Numero\Longitud2` (donde `\Longitud1` y `\Longitud2` pueden ser la misma). Este procedimiento de multiplicación tiene la ventaja de que pueden utilizarse

factores no enteros (0.5, 1.25, etc...). En el caso de que `\Longitud1` sea elástica, la acción anterior la transforma automáticamente en una rígida; por ejemplo, tras:

```
\LongElastica=10pt plus 1fill minus 2fill
\LongElastica=2.5\LongElastica
```

`\the\LongElastica` da como resultado: 25.0pt

El siguiente ejemplo ilustra cómo manejar contadores, definiendo un nuevo comando `\hora` que calcula la hora, a partir del contador `\time`, que almacena el número de minutos después de la medianoche (probar como ejercicio que el comando funciona correctamente):

```
\def\hora{\newcount\horas  \newcount\minutos
% (Definimos dos nuevos contadores)
\horas=\time \global\divide\horas by 60
% (la parte entera de la division produce la hora)
 \minutos=\horas \multiply\minutos by 60
 \advance\minutos by -\time
 \global\multiply\minutos by -1
% (multiplicamos las horas por 60, restamos \time,
% y cambiamos de signo para obtener los minutos)
 \the\horas:\ifnum\minutos<10 0\fi\the\minutos}
% (se imprime horas:minutos, con un cero extra si minutos < 10)
```

4.6. Manejo de cajas en TEX

Cuando TEX compone un documento, trabaja manejando diversos objetos como si fuesen cajas con tres diferentes dimensiones (altura, anchura y profundidad), medidas con respecto a un punto de referencia. Por ejemplo, las líneas se componen alineando las cajas asociadas a cada carácter, según la línea base. Entonces, cada línea se convierte a su vez en una caja, que se alinea (ahora verticalmente), y así sucesivamente hasta que se construye la página.

Existen tres modos fundamentales de trabajo de TEX, a la hora de componer cajas:

- **Modo horizontal:** TEX agrupa cajas alinéandolas horizontalmente unas junto a otras, a lo largo de la línea base, creando una nueva caja de anchura igual a la suma de anchuras, y de altura y profundidad iguales a la mayor de las alturas y profundidades de las cajas, respectivamente. Existen dos sub-modos diferentes dentro del modo horizontal:
  - Ordinario: Es el característico cuando se construyen párrafos; se alinean caracteres horizontalmente, y después se va cortando para formar líneas de la misma anchura. TEX estira ó contrae los espacios para optimizar el resultado final
  - Restringido: En este modo, sólo se alinean las cajas horizontalmente, sin posibilidad de dividir la caja resultante en cajas más pequeñas. Dentro de este modo, no se entienden los comandos asociados a saltos de línea, párrafo, etc...
- **Modo vertical:** Se agrupan las cajas verticalmente unas sobre otras (manteniendo los puntos de referencia en la misma vertical), creando una caja con anchura igual

a la mayor de las anchuras de las subcajas, y con altura total (suma de altura y profundidad) igual a la suma de alturas y profundidades de las subcajas. Al igual que para el modo horizontal, existen dos sub-modos:

  - Ordinario: Es el modo por defecto, en el cual TEX va recogiendo todas las cajas creadas en los modos horizontal y matemático, para empaquetarlas verticalmente.
  - Interno: Se limita a apilar verticalmente cajas, creando una caja indivisible (por ejemplo, cuando se construyen las columnas de una tabla ó matriz)

- **Modo matemático:** Se abre para escribir símbolos ó fórmulas matemáticas, existe en dos variantes, *ordinario* (ó tipo párrafo) y *resaltado* (para fórmulas centradas y resaltadas), las cuales ya se han descrito en el capítulo correspondiente

En los sucesivo, discutiremos algunos comandos de TEX útiles para crear y colocar cajas (que por supuesto, pueden ser utilizados dentro de cualquier documento LATEX; al fin y al cabo, LATEX, como ya se ha repetido, no es más que un conjunto de macros construidas a partir de TEX, que es el lenguaje de bajo nivel que realmente compila el documento fuente).

4.6.1. Cajas horizontales; `\hbox`

El comando `\hbox{Material}` crea cajas indivisibles, en las que el contenido se escribe de izquierda a derecha. El argumento `Material` es procesado *en modo horizontal restringido*, y puede estar compuesto de varias cajas. En realidad, el comando `\mbox` de LATEX no es más que `\hbox`:

```
\def\mbox#1{\leavevmode\hbox{#1}}
```

(el comando `\leavevmode` se asegura de salir del modo vertical, si estuviésemos dentro de él).

Cada caja creada mediante `\hbox` tiene una anchura natural dependiente de la anchura del `Material` incluido en ella. Al igual que ocurría con el comando `\makebox`, es posible cambiar dicha anchura a nuestro gusto:

- `\hbox to Ancho{Material}` Crea una caja de anchura `Ancho` y coloca en ella el material de izquierda a derecha. Si la anchura del material es menor que `Ancho`, se estirarán los espacios elásticos para ocupar todo el espacio disponible, mientras que si es mayor, el material sobresaldrá de la caja (con lo que se sobreescribirá en texto que venga a continuación); véase el siguiente ejemplo (donde se añade `\fbox` a fin de remarcar las cajas:

```
\parindent 0pt Normal:
\fbox{\hbox{caja ejemplo}}
(texto) \\ Estiramos:
\fbox{\hbox to 3cm{caja ejemplo}}
(texto) \\ Contraemos: \fbox{\hbox
to 1cm{caja ejemplo}} (texto) \\
```

Normal: caja ejemplo (texto)
Estiramos: caja ejemplo (texto)
Contraemos: caja ejemplo(texto)

- `\hbox spread Ancho{Material}` Es análogo al anterior, con la diferencia de que aumenta (ó disminuye si el valor es negativo) la anchura natural de la caja en la cantidad `Ancho`:

```
\parindent 0pt
Estiramos: \fbox{\hbox spread
5mm{caja ejemplo}} (texto) \\
Contraemos: \fbox{\hbox spread
-5mm{caja ejemplo}} (texto) \\
```

Estiramos: caja ejemplo (texto)
Contraemos: caja ejemplo (texto)

Con lo ya visto, podemos ahora entender la definición de los comandos `\leftline`, `\centerline`, `\rlap`, etc..., que muestran la potencia de combinar las manipulaciones de cajas y longitudes:

```
\def\leftline#1{\hbox to \hsize{#1\hss}}
\def\rightline#1{\hbox to \hsize{\hss#1}}
\def\centerline#1{\hbox to \hsize{\hss#1\hss}}
\def\rlap#1{\hbox to 0pt{#1\hss}}
\def\llap#1{\hbox to 0pt{\hss#1}}
```

donde recordemos que el comando `\hss` equivale a `\hspace{0pt plus 1fil minus 1fil}`; la longitud `\hsize` almacena la anchura del texto: normalmente equivale a `\textwidth`, aunque puede modificarse a voluntad.

Ejercicio:

Ponemos dos palabras en una caja Test Test de forma que haya un centímetro de separación con los extremos, y dos centímetros de separación entre las palabras.

4.6.2. Cajas verticales; `\vbox`

Las cajas verticales se construyen con el comando `\vbox{Material}`; este comando inicia el modo vertical interno, aunque es posible que TEX ya esté en ése modo antes de invocar el comando. El comportamiento del comando depende de si la caja vertical contiene texto en el nivel más alto, o si contiene el comando `\vrule` (explicado más adelante); en ambos casos la anchura será la de una línea de texto (`\hsize`). Veamos algunos ejemplos:

```
\parindent 0pt \fbox{\vbox{Texto de
prueba \hbox{Una caja} \hbox{Otra
caja}}} \fbox{\vbox{\hbox{Una
caja} Texto de prueba \hbox{Otra
caja}}} \fbox{\vbox{\hbox{Una caja}
\hbox{Otra caja} \hbox{Otra caja mas}}}
\fbox{\vbox{\hsize 4cm Texto de prueba
\hbox{Una caja} \hbox{Otra caja}}} \par
% Nótese la diferencia entre
% modos horizontal y vertical:
\fbox{\vbox{Texto de prueba}} \par
\fbox{\vbox{\hbox{Texto de prueba}}}
```

Texto de prueba Una caja Otra caja

Una caja
Texto de prueba Otra caja

Una caja
Otra caja
Otra caja mas

Texto de prueba
Una caja Otra caja

Texto de prueba

Texto de prueba

Al igual que con las cajas horizontales, se puede fijar de antemano la altura de una caja vertical con:

`\vbox to Alto{Material}` ó `\vbox spread Alto{Material}`

cuyo significado es idéntico a lo ya visto para `\hbox` (cambiando Ancho por Alto).

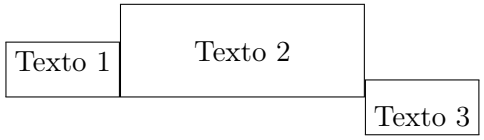
Es interesante remarcar la flexibilidad y potencia del comando `\vbox` de TEX; a diferencia de lo que ocurre con `\hbox`, no existen realmente comandos en LATEX con tanta capacidad; por ejemplo, `\parbox` ó el entorno `minipage` permiten fijar la altura de la caja, pero además requieren fijar su anchura, lo cual puede ser un inconveniente.

El comando `\vbox to Alto{Material}` alinea la línea base de la caja global con la línea base de *la última* caja (es decir, la inferior) incluida en la caja vertical. Existen también los comandos:

`\vtop to Ancho{Material}` y `\vcenter to Ancho{Material}`

que alinean, respectivamente, la parte superior y central de la caja total con la línea base (ATENCIÓN: `\vcenter` sólo se puede emplear dentro del modo matemático).

```
\parindent 0pt
\fbox{\vbox to 5mm{\hbox{Texto 1}}}
\fbox{\vbox to 10mm{\hsize 3cm%
\vfil\centerline{Texto 2}\vfil}}
\fbox{\vtop to 5mm%
{\vfil\hbox{Texto 3}}}
```



4.6.3. Moviendo cajas

Dependiendo del modo (horizontal ó vertical) en el que nos encontremos, disponemos de diversos comandos para desplazar cajas. *En el modo horizontal*, podemos desplazar cajas verticalmente con:

`\raise` Desplazamiento ó `\lower` Desplazamiento

donde Desplazamiento es cualquier longitud. De hecho, éstos dos comandos son esencialmente el mismo, ya que `\raise D` = `\lower -D`. Al usar éstos comandos, la línea base queda inalterada, aunque la altura y profundidad pueden cambiar. La nueva altura y profundidad de la caja se calculan dependiendo de los desplazamientos. Véase el siguiente ejemplo, donde se remarca la línea base con el comando `\hrule` (descrito a continuación):

```
Texto de prueba ; Texto de prueba ; Texto de prueba ; Texto de prueba \\
\fbox{\hbox{\hbox to 0pt{\vbox{\hrule width 6cm}} \hbox{pepe}
\lower3mm\hbox{pepe} \hbox{pepe}}} \ andres \ jaime \hspace{5mm}
\fbox{\hbox{\hbox to 0pt{\vbox{\hrule width 6cm}} \hbox{pepe}
\raise5mm\hbox{pepe} \hbox{pepe}}} \ andres  \ jaime \\
Texto de prueba ; Texto de prueba ; Texto de prueba ; Texto de prueba
```

Texto de prueba ; Texto de prueba ; Texto de prueba ; Texto de prueba

pepe pepe andres jaime pepe pepe andres jaime
pepe pepe

Texto de prueba ; Texto de prueba ; Texto de prueba ; Texto de prueba

---

El comando `\kern Longitud` se utiliza con carácter general para desplazar cajas una cantidad `Longitud` (que puede ser negativa). La dirección del desplazamiento, horizontal

ó vertical, depende que en que modo esté TEX trabajando; en el modo horizontal (en una caja `\hbox`) el desplazamiento es horizontal, mientras que en el modo vertical (en una caja `\vbox`) el desplazamiento es vertical. Veamos como ejemplo el código TEX para obtener el logotipo "TEX":

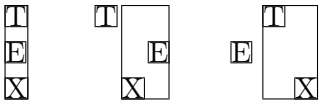
```
\hbox{T\kern-.1666em\lower.5ex\hbox{E}\kern-.125ex X}
```

Para mover horizontalmente las componentes de una caja vertical `\vbox` se utilizan los comandos:

`\moveleft Desplazamiento` y `\moveright Desplazamiento`

Es interesante hacer notar que la anchura de la caja tras los desplazamientos sólo se modifica con desplazamientos a la derecha, y no a la izquierda: la anchura se calcula comenzando en el punto de referencia y extendiéndose hacia la derecha hasta la parte derecha de la componente más alejada; por ejemplo (nótese cómo ahora se utiliza `\frame` para remarcar las cajas, en vez de `\fbox`, que dejaba un pequeño espacio `\fboxsep` alrededor):

```
\frame{\vbox{\hbox{\frame{T}}%
\hbox{\frame{E}}\hbox{\frame{X}}}}
\hspace{1cm}\frame{\vbox{\moveleft10pt%
\hbox{\frame{T}}\moveright10pt\hbox%
{\frame{E}}\hbox{\frame{X}}}}%
\hspace{1cm}
\frame{\vbox{\hbox{\frame{T}}%
\moveleft12pt\hbox{\frame{E}}%
\moveright12pt\hbox{\frame{X}}}}
```



4.6.4. Modificando, creando y reutilizando cajas

Hemos visto ya el modo de guardar y reutilizar cajas en LATEX; veremos ahora el modo, más general, de manipularlas a través de TEX. Podemos declarar una caja nueva con el comando: `\newbox\NombreCaja`, tras lo cual se almacena una caja en la variable `\NombreCaja` con:

`\setbox\NombreCaja=Caja`

A diferencia del comando `\sbox`, que sólo maneja cajas horizontales, con el comando `\setbox Caja` puede ser tanto horizontal como vertical.

Otra forma de declarar cajas es hacer uso de los registros (256) de los que TEX dispone para guardar cajas. Están numerados de 0 a 255, estando el número 255 reservado para la caja de la página. Podríamos entonces, en vez de declarar primero `NombreCaja` con `\newbox`, crear directamente cajas numeradas con:

`\setbox1=Caja1` `\setbox2=Caja2` etc...

(tras lo cual, para todos los comandos descritos a continuación, se debería reemplazar `\NombreCaja` por 1, 2, ...)

Para insertar dentro de un documento los contenidos de una caja, se emplean los comandos:

`\box\NombreCaja` Tras ser usado, borra el contenido de la caja
`\copy\NombreCaja` Usa el contenido de la caja sin borrarlo

Por ejemplo: `\setbox1=\hbox{A} \fbox{\box1} \fbox{\box1}` produce $\boxed{\text{A}}\,\square$, mientras

que `\setbox1=\hbox{A} \fbox{\copy1} \fbox{\copy1}` produce [A][A]

Para una caja `\NombreCaja` dada, las siguientes *longitudes* almacenan, respectivamente, los valores de anchura, altura y profundidad de la caja:

`\wd\NombreCaja` `\ht\NombreCaja` `\dp\NombreCaja`

Véase el siguiente ejemplo; definimos: `\newbox\NuevaCaja` y asignamos `\setbox\NuevaCaja=\hbox{A B C}` tras lo cual, `\the\wd\NuevaCaja` produce 30.99307pt (la anchura de la caja). Podemos "estirar" la caja con: `\wd\NuevaCaja=2\wd\NuevaCaja`, tras lo cual `\frame{\copy\NuevaCaja}` produce [A B C]

Los siguientes comandos son análogos a `\box` y `\copy`, pero, en vez de simplemente escribir el contenido de la caja, las "desmembran" en sus subcomponentes en el momento de ser usadas. Hay versiones horizontal y vertical, así como versiones "`\box`" y "`\copy`", que respectivamente vacían ó no la caja tras ser usada:

`\unhbox\NombreCaja` `\unvbox\NombreCaja`
`\unhcopy\NombreCaja` `\unvcopy\NombreCaja`

El siguiente ejemplo ilustra la diferencia entre simplemente copiar una caja, y desmembrarla con `\unhbox`:

```
\setbox1=\hbox{A B}
\setbox2=\hbox to 2.0\wd1{\unhcopy1}
\frame{\copy2}
\setbox3=\hbox{A B}
\setbox4=\hbox to 2.0\wd3{\copy3}
\frame{\copy4}
```

[A B] [A B]

en el primer caso, tras desmembrar la caja, al construir una caja de anchura doble a la primitiva los elementos se reparten tratando de llenar toda la caja; en el segundo, al estar la caja intacta, los elementos se mantienen a la derecha de la caja `\box4`, que contiene a `\box3`

4.6.5. Rayas horizontales y verticales

En TEX se pueden utilizar dos tipos de rayas ó cajas negras; las horizontales, `\hrule`, y las verticales, `\vrule`. Para cada una de ellas se pueden especificar tres dimensiones: anchura, altura y profundidad:

`\hrule height` Altura `width` Anchura `depth` Profundidad
`\vrule height` Altura `width` Anchura `depth` Profundidad

puede omitirse cualquiera de estos tres parámetros, en cuyo caso TEX asignará valores por defecto:

- Altura 0.4 pt y profundidad 0 pt, si la raya es horizontal (`\hrule`)
- Anchura 0.4 pt, si la raya es vertical (`\vrule`)
- El resto de dimensiones se obtiene extendiendo la raya indefinidamente hasta completar el tamaño de la caja que la contiene

La diferencia esencial entre `\hrule` y `\vrule` reside en que `\hrule` es material *vertical*, por lo que sólo puede ser utilizado entre párrafos ó dentro de una caja vertical `\vbox`, mientras que `\vrule` es material *horizontal*, por lo que sólo puede utilizarse dentro de un párrafo ó de una caja horizontal `\hbox`.

Ejemplos:

```
\hbox{Ejemplo \vrule width 2pt\vbox to
25pt{linea \par vertical}} \vspace{3mm}
\vbox{\hbox to 4cm{Otro ejemplo}
\kern 1mm\hrule height 1pt\kern 1mm
\hbox to 3cm{linea horizontal}}
```

Ejemplo linea vertical

Otro ejemplo

linea horizontal

```
\hbox{\vbox{\hbox to 25mm{\hfil%
\hbox{Texto 1}\hfil}\kern2pt\hrule}%
\vrule \lower7.5mm\vbox to 15mm{\hrule%
\kern-11pt\hbox to 25mm{\hfil%
\hbox{Texto 2}\hfil}\vfil\hbox to
25mm{\hfil\hbox{Texto 3}\hfil}%
\kern2pt\hrule}}
```

Texto 2

Texto 1

Texto 3

4.7. Repetición de objetos

Veremos ahora otro modo de repetir objetos, ligeramente diferente del comando `\multiput` ya visto. En vez de proporcionar el número de objetos a repetir, puede interesarnos llenar un cierto espacio, de longitud fija ó variable, con copias de un objeto. Mediante el comando `\leaders` se pueden obtener copias de un objeto en tal forma. Para ello, debemos especificar el objeto a copiar y el espacio que debe ser completado con copias de tal objeto. La sintaxis del comando es la siguiente:

`\leaders` *Objeto* `\hskip` *Longitud*

donde hay que tener en cuenta que *Objeto* debe ser una caja, y *Longitud* puede ser cualquier longitud (incluyendo elásticas). Se puede reemplazar `\hskip 1fil` por simplemente `\hfil`, ó `\hskip 1fill` por `\hfill`. Por ejemplo:

```
\noindent\null\leaders\hrule\hfill
\null\\[2mm]
\null\leaders\hbox{/\textbackslash}%
\hskip.4\hsize\null\\[2mm] \hbox to
4cm{\leaders\hbox{\frame{\hbox to
10pt{\vbox to 10pt{}}} }\hfill}
```

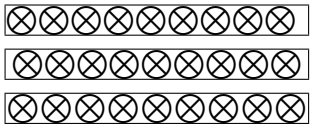
Es importante tener en cuenta que es necesario "marcar" los puntos entre los que actúa el comando `\leaders`, sobre todo si estamos utilizando longitudes elásticas. Es por eso que se utiliza en el ejemplo anterior `\null` (otras posibilidades equivalentes serían `\mbox{}` ó `\kern0pt`).

En el caso de que el objeto a repetir no sea un múltiplo entero del tamaño de la caja que contiene a las copias del objeto, aparecerá cierta asincronía. Para solucionar esto, se dispone de otras dos variantes de `\leaders` para repetir un objeto:

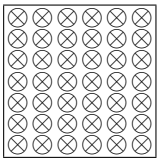
`\cleaders` *Objeto* `\hskip` *Longitud*
`\xleaders` *Objeto* `\hskip` *Longitud*

mientras que `\leaders` aparta el espacio sobrante a la derecha, `\cleaders` reparte el espacio sobrante a ambos lados (centrando las copias del objeto en la caja); `\xleaders` distribuye el espacio sobrante entre cada copia del objeto, ajustando las copias del objeto a la caja; por ejemplo:

```
\frame{\hbox to 4cm{\leaders%
\hbox{$\bigotimes$}\hfill}}\par
\frame{\hbox to 4cm{\cleaders%
\hbox{$\bigotimes$}\hfill}}\par
\frame{\hbox to 4cm{\xleaders%
\hbox{$\bigotimes$}\hfill}}
```



De igual manera a lo ya visto, se pueden repetir objetos en la dirección vertical; para ello, simplemente se cambia el segundo argumento del comando `\leaders` de horizontal (`\hskip` *Longitud*) a vertical (`\vskip` *Longitud*). Como ejercicio, escribir el código TEX para obtener:



(la clave está en repetir el objeto `$\otimes$` primero horizontalmente, y repetir la repetición verticalmente a continuación)

4.8. Sistematizando tareas

Veremos ahora una serie de comandos útiles para sistematizar tareas; imaginemos que cada vez que se inicia un párrafo, fórmula, caja, ..., queremos que se ejecuten una serie de comandos. Para ello, se tienen las siguientes posibilidades:

• `\everypar{Comandos}` Antes de empezar a dar formato a cada párrafo, se ejecutan `Comandos`. Veamos un ejemplo, en el que definimos un nuevo contador `parrafo`, y para cada párrafo, se le pone como título "Párrafo n" en negrita y centrado:

```
\parindent 0pt \parskip 10pt
\setcounter{parrafo}{0}
\everypar{\addtocounter{parrafo}{1}
\centerline{\bfseries Párrafo %
\theparrafo}\\[1mm]}
Esto es un primer párrafo de
ejemplo; en el código anterior...\par
...se puede ver cómo incrementamos
el valor del contador \texttt{parrafo}
al empezar cada párrafo...\par ...y lo
recuperamos, para imprimirlo, con el
comando \verb+\theparrafo+\par MUY
IMPORTANTE: Nótese cómo, en este
documento, el comando
\verb+\everypar{...}+ es local; al
estar incluido dentro de un
entorno multicols (utilizado para
escribir este ejemplo) no tiene
efecto al salir del entorno
```

Párrafo 1

Esto es un primer párrafo de ejemplo; en el código anterior...

Párrafo 2

...se puede ver cómo incrementamos el valor del contador `parrafo` al empezar cada párrafo...

Párrafo 3

...y lo recuperamos, para imprimirlo, con el comando `\theparrafo`

Párrafo 4

MUY IMPORTANTE: Nótese cómo, en este documento, el comando `\everypar{...}` es local; al estar incluido dentro de un entorno multicols (utilizado para escribir este ejemplo) no tiene efecto al salir del entorno

• `\everymath{Comandos}` Análogo a `\everypar`; se ejecutan los comandos cada vez que entremos en modo matemático *ordinario* (ó modo texto)

• `\everydisplay{Comandos}` En este caso, los comandos se ejecutan cada vez que se abre el modo matemático *resaltado*. Imaginemos que queremos que todas las fórmulas resaltadas se escriban en color rojo; para conseguirlo, simplemente se puede declarar: `\everydisplay{\color{red}}`

• `\everyhbox{Comandos}` y `\everyvbox{Comandos}` Ejecutan los comandos cada vez que comienze una caja horizontal (`\hbox`) ó vertical (`\vbox`), respectivamente.

4.9. Condicionales y bucles

El compilador TEX posee amplias capacidades a la hora de programar diversas acciones. Además de poder manejar diversos registros (contadores, longitudes, cajas) con total flexibilidad, su potencia se ve reforzada al ser posible incluir bucles y condicionales dentro de un documento.

Un condicional es una estructura de control que elige entre diversas acciones en función del valor de una variable lógica; su forma general es:

`IF` <Test> [Instrucciones A] `ELSE` [Instrucciones B] `END IF`

lo cual significa que, de cumplirse la condición <Test>, se ejecutarán las instrucciones "A", y de no cumplirse, las instrucciones "B". En lenguaje TEX, el condicional se escribe:

```
\if<Test> [Parte A] \else [Parte B] \fi
```

aunque podemos prescindir de cualquiera de las partes (A ó B), y tener simplemente:

```
\if<Test> [Parte A] \fi      ó      \if<Test> \else [Parte B] \fi
```

(el último, correspondería a una versión "de negación" del condicional).

Los condicionales pueden anidarse sin problemas; cada `\fi` se asume que corresponde con el más reciente `\if`. A continuación describiremos algunos de los 17 condicionales que están definidos en TEX, correspondientes a diversos formatos de la condición <Test>;

• `\ifnum Número1 Relación Número2`

Se utiliza para comparar números enteros, con `Relación` igual a <, >ó =. Como ejemplo, definimos un contador `\cuatrodigit`, que imprime números es formato de cuatro dígitos, con independencia de su tamaño:

```
\def\cuatrodigit#1{%
\ifnum #1<1000 0\fi
\ifnum #1<100 0\fi
\ifnum #1<10 0\fi #1}
```

tras lo cual,
`\cuatrodigit{8} - \cuatrodigit{18} - \cuatrodigit{198} - \cuatrodigit{1238}`
produce:
0008 - 0018 - 0198 - 1238

• `\ifodd Número`

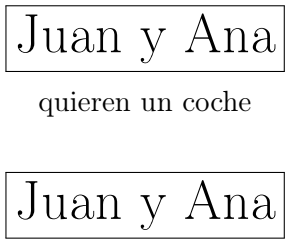
sirve para comprobar si un número entero es impar. En el caso de que queramos analizar el valor de un determinado contador, recordemos que debemos sustituir `Número` por `\value{NombreContador}`, si estamos trabajando con un contador definido en LATEX. Por contra, si el contador ha sido definido en TEX (con `\newcount`), podemos recuperar el valor numérico con `\the\NombreContador` ó `\number\NombreContador`

Ejemplo: Compilando `Esta página es \iffodd\value{page} impar \else par\fi` obtendremos "Esta página es impar", si es impar, ó "Esta página es par" si es par.

- `\ifdim Dimensión1 Relación Dimensión2`

se utiliza para comparar dos longitudes. Como ejemplo, vamos a construir un comando que crea una caja enmarcada con un texto en tamaño `\huge` (que será el primer argumento del comando) y un texto de leyenda, que se colocará centrada si la longitud de la leyenda es menor que la del texto principal, ó en estilo párrafo si es mayor:

```
\newlength{\anchura}
\def\textoresaltado#1#2{%
\setbox1=\hbox{\fbox{\huge#1}}
\settowidth{\anchura}{#2}\vbox{\copy1%
\vspace{6pt}\ifdim\anchura<\wd1\hbox
to\wd1{\hss#2\hss}\else%
\hbox{\parbox{\wd1}{#2}}\fi}}
\textoresaltado{Juan y Ana}{quieren
un coche}\par\medskip
\textoresaltado{Juan y Ana}{necesitan
comprarse un coche nuevo porque el
antiguo se les ha quedado viejo}
```



- `\ifhmode` • `\ifvmode` • `\ifmmode`

sirven para comprobar, respectivamente, si estamos dentro del modo horizontal, vertical, ó matemático (en cada caso, no se distingue entre los diferentes sub-modos). Por ejemplo, el comando `\ensuremath` de LaTeX está definido como:

```
\newcommand{\ensuremath}[1]{\ifmmode #1\else $#1$\fi}
```

- `\ifcase Numero [Caso n=0] \or [Caso n=1] \or ...`
  `[Caso n=M] \else [Caso n=Otro Numero] \fi`

sirve para ejecutar diferentes acciones, de acuerdo a los valores que tome la variable `Numero` (que puede, por ejemplo, ser un contador); si n=0 se ejecutaran las primeras instrucciones, si n=1 las, segundas, y así sucesivamente hasta M; opcionalmente podemos colocar más instrucciones después de `\else`, que se ejecutarán si `Numero` es menor que 0 ó mayor que M. Véase el siguiente ejemplo, que traduce números naturales a notación hexadecimal:

```
\def\hexadec#1{\ifcase #1 %
0\or 1\or 2\or 3\or 4\or %
5\or 6\or 7\or 8\or 9\or %
A\or B\or C\or D\or E\or F\fi}
```

Comando	Resultado
`\hexadec{7}`	7
`\hexadec{12}`	C

- `\ifx Argumento1Argumento2`

compara dos argumentos entre sí, siendo verdadero si son iguales y falso si son distintos. Argumento1 y Argumento2 pueden ser caracteres, cajas, comandos... Es importante puntualizar que, a la hora de comparar cadenas de caracteres ó cajas, es necesario con anterioridad incluir tales contenidos en sendos comandos, que serán después comparados. Por ejemplo:

```
\columnbreak \vspace*{0.1cm}
\def\a{Hola} \def\b{Hola} \def\c{hola}
\def\d{H} \def\e{\hbox{hola}}
\def\f{\hbox{hola}}   Comparación 1:
\ifx AA iguales \else distintos \fi
% (caracteres aislados son comparables)
Comparación 2:
\ifx \a\b iguales \else distintos \fi
Comparación 3:
\ifx \a\c iguales \else distintos \fi
Comparación 4:
\ifx H\d iguales \else distintos \fi
Comparación 5:
\ifx \c\e iguales \else distintos \fi
Comparación 6:
\ifx \e\f iguales \else distintos \fi
```

Comparación 1: iguales
Comparación 2: iguales
Comparación 3: distintos
Comparación 4: distintos
Comparación 5: distintos
Comparación 6: iguales

Pregunta de trivial: ¿Por qué en el primer caso hay un espacio extra?

LaTeX también proporciona algunos condicionales predefinidos, que es bueno conocer:

- `\if@twoside`
- `\if@twocolumn`

son verdaderos si se está procesando el documento con las opciones twoside ó twocolumn, respectivamente, y falsos en caso contrario.

- `\@ifnextchar Carácter{ParteA}{ParteB}`

Se procesa `ParteA` en caso de que el siguiente carácter coincida con `Carácter`, y `ParteB` en caso contrario. Este condicional es muy utilizado en LaTeX en los comandos que utilizan argumentos opcionales, caracterizados por ir entre corchetes. Veamos un ejemplo de cómo definir un comando con dos argumentos, uno de ellos optativo; queremos recuadrar un texto dado con una línea de grosor variable, 0.4 pt por defecto:

```
\def\mirecuadro[#1]#2{{\fboxrule#1\fbox{#2}}}
\makeatletter
\def\recuadro{\@ifnextchar[{\mirecuadro}{\mirecuadro[0.4pt]}}
\makeatother
```

Tras esto, `\recuadro{Prueba}` resultará en Prueba, mientras que si queremos cambiar el grosor de línea a 1pt, deberemos escribir: `\recuadro[1pt]{Prueba}` $\longrightarrow$ Prueba. ¿Por qué los comandos `\makeatletter` y `\makeatother`? La razón está en que, por defecto, está prohibido utilizar el símbolo @ en los comandos dentro de un documento; éste comando se utiliza frecuentemente en las clases de documentos ó paquetes, por lo que se restringe su uso para evitar coincidencias casuales con comandos ya definidos. El comando `\makeatletter` levanta esta prohibición, mientras que el comando `\makeatother` la vuelve a recuperar.

Podemos utilizar lo aprendido en el ejemplo anterior para construir comandos más complicados. Por ejemplo, modifiquemos el comando `\recuadro` (renombrándolo a `\Recuadro`) para que ahora admita dos argumentos optativos, según la sintaxis:

`\Recuadro[Grosor](Color){Texto}`

siendo `Grosor` la anchura del recuadro (0.4pt por defecto), y `Color` su color (rojo por defecto). Utilizando recursivamente el condicional `\@ifnextchar` se obtiene el resultado deseado (comprobar):

```
\makeatletter
```

```
\def\Mirecuadro(#1)#2{{\color{#1}\fbox{\color{black}#2}}}
\def\Mirecuadroaux[#1]{\fboxrule#1\@ifnextchar(%
{\Mirecuadro}{\Mirecuadro(red)}}
\def\Recuadro{\@ifnextchar[{\Mirecuadroaux}{\Mirecuadroaux[0.4pt]}}
\makeatother
```

4.9.1. Nuevos condicionales

Volviendo a TEX, veremos ahora la forma de definir nuevos condicionales con el comando `\newif`, de sintaxis:

`\newif\if`*Nombre*

donde *Nombre* corresponderá al nombre del nuevo condicional. El comando `\newif` se encarga de definir tres nuevos comandos:

- `\`*Nombre*`true` Asigna a la variable lógica *Nombre* el valor verdadero
- `\`*Nombre*`false` Asigna a la variable lógica *Nombre* el valor falso
- `\if`*Nombre*`...` `\else`... `\fi` Nuevo condicional, que ejecuta una acción u otra según el valor que se le haya asignado anteriormente a la variable lógica *Nombre*

Como ejemplo, definamos un nuevo entorno `ocultar`, de forma que el texto dentro de tal entorno se muestre ó no en el documento final, dependiendo del valor de una variable lógica:

```
\newbox\boxocultar
\newif\ifocultar
\newenvironment{ocultar}
 {\setbox\boxocultar\vbox\bgroup}
 {\egroup\ifocultar\else\par\unvbox\boxocultar\fi}
```

tras esta definición, si se coloca el comando `\ocultarfalse`, todo el texto dentro de entornos `ocultar` que estén a continuación de este comando no se verá en el documento final; en cambio, sustituyéndolo por `\ocultarfalse`, se reestablecerá el texto dentro de tales entornos, por ejemplo:

```
\ocultartrue \begin{ocultar}
texto de prueba que no se ve
\end{ocultar}
\ocultarfalse \begin{ocultar}
texto de prueba que si se ve
\end{ocultar}
```

texto de prueba que si se ve

La utilidad de éste entorno puede estar, por ejemplo, en la inclusión de notas y comentarios que puede convenirnos suprimir en el documento final; añadir un comando `\ocultartrue` es más rápido que comentar líneas una por una. Merece la pena analizar un poco la definición del nuevo entorno:

1) `\newbox\boxocultar` → Define una nueva caja para almacenar el texto oculto

2) `\setbox\boxocultar\vbox\bgroup` → Abre una caja vertical y la almacena en `\boxocultar`; nótese el empleo del comando `\bgroup`: éste comando es análogo a " { ",

es decir, es un delimitador de grupo. La sutileza radica en que, de usar directamente {, habría un conflicto con la sintaxis del comando `\newenvironment`

3) `{\egroup\ifocultar\else\par\unvbox\boxocultar\fi}` → Tras haber abierto la caja vertical, y haberse rellenado con todo el texto dentro del entorno, se cierra con `\egroup` (análogo a " } "); recordemos que todo lo que iba entre el primer conjunto de llaves corresponde a las instrucciones LaTeX a ejecutar *al entrar en el entorno*, mientras que este segundo conjunto de instrucciones corresponde a lo que debe hacerse *al salir del entorno*. Tras eso, se comprueba con `ifocultar` si el texto debe ocultarse, en cuyo caso, no se hace nada, y, en caso contrario, se deshace e imprime la caja `\boxocultar`

4.9.2. Bucles

Se realizan bucles con el comando:

`\loop ParteA \if... ParteB \repeat`

donde `ParteA` y `ParteB` son conjuntos de comandos, y `\if` es cualquier condicional, sin la correspondiente partícula `\fi`. TeX procesa primero `ParteA`; si la condición es verdadera, procesa `ParteB`, y repite el proceso comenzando de nuevo por `ParteA`; si no, inmediatamente se sale del bucle. Definamos como ejemplo un comando que imprima los primeros n números naturales:

```
\newcount\minum
\def\numeros#1{\ifnum#1< 1%
 \else 1\minum=1\loop
 \advance\minum by 1%
 \ifnum\minum<#1,
 \the\minum\repeat\fi}
\numeros{40}
```

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40

En LaTeX, están predefinidos bucles asociados a condicionales específicos, útiles para manejar números y longitudes:

- `\@whilenum{TestNum} \do {Acción}`
- `\@whiledim{TestLong} \do {Acción}`

En ellos, se evalúa la relación numérica `TestNum` ó `TestLong` (comparación de números ó longitudes, respectivamente); *mientras sea verdadera* se procesarán las instrucciones en `Acción`, terminando el bucle en el momento en que la relación sea falsa. El siguiente ejemplo calcula la sucesión de todos los números *pares* menores que uno dado:

```
\newcount\cuenta
\makeatletter  \def\pares#1{%
\minum=2\@whilenum\minum<#1\do
{\the\minum, \advance\minum by 2}}
\makeatother
Los números pares menores que 95
son: \pares{95}
```

Los números pares menores que 95 son: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94,

También es posible definir bucles en los que la condición de control sea un condicional `\if`*Nombre* definido en TeX con `\newif`:

- `\@whilesw\if`*Nombre*`\fi{Acción}`

con lo que TEX procesará los comandos de `Acción` hasta que el condicional `\if`*Nombre* sea falso.

Finalmente, se pueden también construir estructuras "for / next", en las cuales se ejecuta una serie de acciones para cada uno de los elementos de una lista. Se utiliza la sintaxis:

```
\@for\Nombre:=\lista\do{Acción}
```

donde `\Nombre` es una variable (que no hace falta definirla previamente) que va almacenando los diferentes elementos de una lista (`\lista`), que debe ser previamente definida con `\def`; los elementos de la lista han de estar separados entre sí por comas. En el siguiente ejemplo, primeramente definimos a través de TEX un comando `\longitud{Palabra}` para contar el número de letras de una palabra, que luego utilizamos para crear un tabla con las longitudes de una lista de palabras almacenadas en `\lista`:

```
\newcount\nna  \def\longitud#1{\nna=0%
\expandafter\contar#1\end\number\nna}
\def\contar#1{%
\ifx#1\end\let\next=\relax
\else\advance\nna by1
\let\next=\contar\fi\next}
La longitud de la frase ''pepe tiene
un coche'' es de \longitud{pepe
tiene un coche} caracteres no blancos
```

La longitud de la frase "pepe tiene un coche" es de 16 caracteres no blancos

```
\def\lista{Pepe,Juan,Andrés,Antonio}
\begin{tabular}{l}
Nombre y longitud \\ \hline
\makeatletter \@for\nombre:=\lista
\do{\hbox to 30mm{\nombre\hss}%
\longitud{\nombre} \\ } \makeatother
\end{tabular}
```

Nombre y longitud	
Pepe	4
Juan	4
Andrés	6
Antonio	7

4.9.3. Otros ejemplos

Invertir una palabra

```
\def\Invertir#1{%
\def\INV{}\INVCAD#1\end\INV}%
\def\INVCAD#1{%
\ifx#1\end\let\next=\relax
\else\CONCAD#1%
\let\next=\INVCAD\fi\next}%
\def\CONCAD#1{\edef\INV{#1\INV}}
\Invertir{Espejo}Espejo\par
Curioso\Invertir{Curioso}
```

ojepsEEspejo
CuriosoosoiruC

Numeros primos

```
\newif\ifprime  \newif\ifunknown %
\newcount\n \newcount\p %
\newcount\d \newcount\a %
\def\primes#1{2,~3 % (#1 is at least 3)
  \n=#1 \advance\n by-2 % n more to go
  \p=5 % odd primes starting with p
  \loop\ifnum\n>0 \printifprime\advance\p by2 \repeat}
\def\printp{, % we will invoke \printp if p is prime
  \ifnum\n=1 \fi
  \number\p \advance\n by -1 }
\def\printifprime{\testprimality \ifprime\printp\fi}
\def\testprimality{{\d=3 \global\primetrue
  \loop\trialdivision \ifunknown\advance\d by2 \repeat}}
\def\trialdivision{\a=\p \divide\a by\d
  \ifnum\a>\d \unknowntrue\else\unknownfalse\fi
  \multiply\a by\d
  \ifnum\a=\p \global\primefalse\unknownfalse\fi}
```

Tras lo cual, `\primes{200}` calcula e imprime los 200 primeros números primos:

2, 3 , 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223