C

Approfondissement

TEX n'est pas seulement un logiciel de composition de document, c'est aussi un langage de programmation qui permet de manipuler des macros de façon puissante. LATEX, basé sur TEX, reprend à son compte ces fonctionnalités en les adaptant pour en harmoniser la syntaxe et les rendre plus robustes.

Après avoir complété ce qui a été dit au chapitre 3 à propos du fonctionnement des macros, nous verrons comment manipuler les longueurs, les compteurs et les boîtes. Quelques packages (calc, ifthen, multido) complètent l'offre de base de LATEX pour former un ensemble complet de programmation.

C.1 Commandes personnelles

L'essentiel, c'est-à-dire les macros sans paramètre ou avec paramètres obligatoires, a été présenté au chapitre 3. Nous allons ici approfondir cette notion en présentant la façon de définir des commandes à paramètres optionnels et des environnements.

C.1.1 Paramètres optionnels

Dans certaines situations, le paramètre d'une commande peut être récurrent et devenir fastidieux à stipuler. LATEX offre la possibilité d'utiliser des arguments optionnels aux commandes. La syntaxe complète de \newcommand est :

```
\newcommand{\⟨macro⟩}[⟨nb-paramètres⟩][⟨val-optionnelle⟩]{⟨définition⟩}
```

Les significations de \⟨*macro*⟩, ⟨*nb-paramètres*⟩ et ⟨*définition*⟩ ont été présentées au chapitre 3, nous n'y reviendrons pas. Le paramètre ⟨*val-optionnelle*⟩ indique que le premier paramètre de la commande \⟨*macro*⟩ est optionnel et que, s'il n'est pas précisé, il prendra cette valeur.

Les accolades entourant\⟨*macro*⟩ ne sont pas obligatoires. Beaucoup d'utilisateurs ne les spécifient pas. C'est ce que nous ferons dans la suite de ce chapitre.

Prenons l'exemple d'une commande \siecle permettant de composer un numéro de siècle sous la forme « xviii^{e} siècle » lorsqu'on indique \siecle{xviii}. En d'autres

termes, la commande \siecle accepte un argument indiquant le numéro de siècle en minuscules. La définition suivante fera le travail demandé en utilisant la commande \ieme définie par l'option french du package babel.

```
\newcommand\siecle[1]{\textsc{#1}\ieme}
```

Cette définition comporte cependant un défaut : elle est incorrecte pour le premier siècle qui se note Ier et non Ie. C'est-à-dire avec \ier au lieu de \ieme. Une solution pour remédier à ce défaut serait d'employer un second paramètre qui indiquerait quelle commande utiliser pour la terminaison. Comme \ieme risque d'être beaucoup plus utilisé que \ier, il peut être intéressant de penser au paramètre optionnel :

C-1

```
\newcommand\siecle[2][\ieme]
           {\textsc{#2}#1}
Du \siecle[\ier]{i} au
\siecle{iii} siècle,
```

Du Ier au IIIe siècle,

L'exemple montre que le paramètre optionnel est toujours le premier, donc celui qui sera noté #1 dans la ⟨*définition*⟩. Il montre également qu'on l'indique de façon explicite en utilisant les crochets, et qu'en son absence, c'est bien la ⟨*val-optionnelle*⟩ qui sera prise en compte. Enfin, le nom de la commande nouvellement créée n'a pas été placé entre accolades à la suite de \newcommand : c'est une habitude partagée par un grand nombre d'utilisateurs de LaTeX mais elle n'est nullement obligatoire, on peut tout à fait stipuler le nom de la nouvelle commande entre accolades.

En réalité, pour cette commande particulière, la technique du paramètre optionnel n'est pas la meilleure. On verra comment améliorer cet exemple à la section C.4.1 page 389 avec des tests.

La commande \newcommand dispose d'une forme étoilée. La syntaxe est strictement la même que celle de la version non étoilée mais la commande \par (ou une ligne vide) est interdite dans les arguments. On peut interdire les arguments sur plusieurs paragraphes parce qu'on considère qu'il s'agit d'une erreur (c'est le cas des commandes de sectionnement par exemple). Un autre intérêt est qu'en cas d'oubli de l'accolade fermante, l'erreur sera mieux circonscrite et n'empêchera pas le reste du document d'être compilé correctement.

La macro \newcommand vérifie que la commande en cours de définition n'existe pas déjà. C'est une protection efficace contre les catastrophes. Par exemple, on pourrait vouloir définir une commande \def qui servirait à composer des définitions. Or, \def existe déjà : il s'agit d'une commande de bas niveau servant à définir des commandes. À ce titre, \newcommand se sert justement de \def. Redéfinir \def reviendrait à casser la commande \newcommand ainsi que d'autres tout aussi importantes. En termes moins techniques, LaTeX deviendrait complètement fou !

Cependant, dans certaines circonstances, on peut être amené à redéfinir une commande déjà existante. Dans ce cas, il faut employer \renewcommand. La syntaxe est exactement la même que celle de \newcommand, et il existe également une forme étoilée. Ici, la commande doit obligatoirement exister au préalable.

Signalons rapidement la commande \providecommand qui définit une macro exactement de la même façon que \newcommand mais qui ne fera rien si la macro existe déjà. On a vu un exemple d'une définition utilisant \providecommand dans une note de bas de page de la section 13.2.2 page 324.

C

C.1.2 Environnements

L'utilisateur a la possibilité de se créer des environnements personnels. Pour cela, il suffit d'utiliser la commande \newenvironment.

```
\newenvironment{⟨nom-env⟩}[⟨nb-arg⟩][⟨val-opt⟩]{⟨initial⟩}{⟨final⟩}
```

On définit ainsi l'environnement ⟨*nom-env*⟩ qui demande ⟨*nb-arg*⟩ arguments (aucun par défaut), avec, éventuellement, un premier argument optionnel dont la valeur par défaut sera ⟨*val-opt*⟩ lorsque l'environnement sera utilisé.

Lorsqu'on utilise cet environnement avec la syntaxe

```
\begin{⟨nom-env⟩}{⟨arg1⟩}...{⟨argn⟩}
  ⟨corps⟩
\end{⟨nom-env⟩}
```

le code ⟨*initial*⟩ est exécuté en premier, avec les paramètres #1,...,#*n* correspondant aux arguments ⟨*arg*$_1$⟩,...,⟨*arg*$_n$⟩. Le ⟨*corps*⟩ de l'environnement est alors composé. Enfin, le code ⟨*final*⟩ est exécuté.

Attention

Les arguments ne sont disponibles que dans le code ⟨*initial*⟩. Il n'est pas possible de les employer directement dans le code ⟨*final*⟩.

Supposons que l'on veuille réaliser un environnement composant un texte sur un empagement réduit, avec un titre centré en gras au-dessus. La composition sur un empagement réduit s'obtient avec l'environnement quote. On pourrait procéder de la façon suivante :

C-2

```
\newenvironment*{chapeau}[1]
  {\begin{center}
   \normalfont\bfseries #1
   \end{center}
   \begin{quote}
  }
  {\end{quote}}
```

```
\begin{chapeau}{Résumé}
  Ce chapitre montre les intérêts
  de la programmation sous \LaTeX.
\end{chapeau}
\section{Introduction}
Après avoir étudié la façon de
définir de nouvelles commandes et
de nouveaux environnements,\dots
```

Résumé

Ce chapitre montre les intérêts de la programmation sous LaTeX.

1 Introduction

Après avoir étudié la façon de définir de nouvelles commandes et de nouveaux environnements,...

Dans cet exemple, lorsque LaTeX rencontre le code \begin{chapeau}{Résumé}, il commence par exécuter le code

C

```
\begin{center}
\normalfont\bfseries Résumé
\end{center}
\begin{quote}
```

puis compose le ⟨*corps*⟩ de l'environnement avant d'exécuter le code

```
\end{quote}
```

La commande \normalfont est une précaution pour être certain de revenir aux attributs par défaut de la fonte principale du texte. En effet, on ne sait jamais dans quelles circonstances cet environnement peut être appelé.

L'exemple montre également que c'est la forme étoilée de \newenvironment qui a été utilisée. Son rôle est le même que pour \newcommand, à savoir interdire les paramètres sur plusieurs paragraphes. Il existe enfin une commande \renewenvironment qui permet de définir un environnement qui existe déjà.

C.2 Longueurs et compteurs

LaTeX permet de manipuler des longueurs et des compteurs de façon puissante. Avec le concept de boîte, qui sera vu à la section suivante, ces entités permettent de composer du matériel de façon très variée. C'est par exemple avec de tels calculs que l'on demande à LaTeX de composer un *filet* ayant la largeur d'empagement plus la largeur de la marge de note. Avec le package calc, qui sera présenté page 383, nous verrons des exemples pratiques de tels calculs.

C.2.1 Espaces non standards

Comme on l'a déjà dit, LaTeX permet d'insérer des espaces de longueurs arbitraires et ce, avec une précision que ne permettent pas les traitements de texte. Nous disposons pour cela des deux commandes suivantes dévolues aux espaces respectivement horizontales et verticaux :

```
\hspace{⟨longueur⟩} \vspace{⟨longueur⟩}
```

où ⟨*longueur*⟩ peut être spécifié sous la forme ⟨*nombre*⟩⟨*unité*⟩, où ⟨*nombre*⟩ est un nombre décimal [1] éventuellement négatif et ⟨*unité*⟩ est l'une des unités de longueur reconnues par LaTeX, étudiées à la section suivante (en particulier le tableau C.1 page 378).

C-3

```
J'aime\hspace{1,2cm}mieux forger
mon âme que la\hspace{-2mm}meubler.
```

J'aime mieux forger mon âme
que lmeubler.

L'espace provoquée par la commande \hspace est ignorée si elle chevauche le début ou la fin d'une ligne. La version étoilée \hspace* de cette commande crée une espace qui, elle, n'est jamais ignorée.

1. Le séparateur décimal peut être le point ou la virgule.

Certaines espaces horizontales définies par LaTeX seront étudiées à la section C.2.2. Parmi elles : \quad, \qquad et \hfill. Par exemple, la commande \hfill permet d'espacer de façon régulière les objets qui l'entourent à la manière d'un ressort ; si on souhaite espacer avec des ressorts « pondérés », il suffit de passer, en argument de la commande \hspace, la commande \stretch{⟨*poids*⟩} où ⟨*poids*⟩ est un nombre décimal. Dans l'exemple suivant, les deux phrases occupent chacune une ligne entière avec, dans la première, deux espacements vides de même longueur (\hfill est un raccourci de \hspace{\stretch{1}}) et, dans la deuxième, une seconde espace de longueur trois fois la première.

C-4
```
J'aime\hfill mieux forger mon âme que la\hfill meubler.

J'aime\hspace{\stretch{1}}mieux forger mon âme que
la\hspace{\stretch{3}}meubler.
```
J'aime mieux forger mon âme que la meubler.
J'aime mieux forger mon âme que la meubler.

Certaines commandes, telles que \hspace et \vspace, demandent des longueurs comme paramètres. Ces deux commandes ont été présentées au chapitre 2.

C-5
```
\newlength{\largeur}
\settowidth{\largeur}{Matériel}
*Matériel*\par
*\hspace{0.5\largeur}*\par
*\hspace{\largeur}*\par
*\hspace{2\largeur}*\hspace{-\largeur}+
```
Matériel
* *
* *
* \+ *

Ces espacements disparaissent au niveau des frontières de l'empagement. C'est-à-dire qu'un \hspace n'aura aucune action en début ou en fin de ligne et qu'un \vspace n'aura aucune action en début ou en fin de page. C'est généralement ce qui est souhaité mais si l'on veut passer outre, il suffit d'utiliser la forme étoilée de ces commandes.

C-6
```
\noindent
Paragraphe sans retrait\\
\hspace{1cm}Pas de retrait\\
\hspace*{1cm}Retrait
```
Paragraphe sans retrait
Pas de retrait
Retrait

Attention

L'emploi des commandes \\ et \hspace* pour simuler un nouveau paragraphe est une mauvaise idée.

On dispose cependant pour les espaces verticaux de la commande \vspace, analogue à \hspace. Cette commande a toutefois un comportement inattendu si elle est utilisée à l'intérieur d'un paragraphe : ce n'est pas à l'endroit où elle est insérée qu'est provoqué l'espace vertical, mais à la fin de la ligne formatée par LaTeX qui la contient. Si on souhaite que l'espacement ait lieu exactement là où est insérée la commande,

celle-ci doit être suivie d'un changement de paragraphe. Ainsi, dans l'exemple suivant, le premier espace vertical n'a pas lieu juste après « J'aime », mais à la fin de la ligne produite.

C-7

```
J'aime\vspace{0,8cm} mieux forger
mon âme que la meubler.
\vspace{0,5cm}

J'aime mieux forger mon âme que la
meubler.
```

J'aime mieux forger mon âme que la

meubler.

J'aime mieux forger mon âme que la meubler.

Il existe également une version étoilée `\vspace*` dont l'espace créée n'est pas supprimée si elle se trouve à une coupure de page, contrairement à celle provoquée par la commande `\vspace`.

Les espaces verticaux `\smallskip`, `\medskip`, `\bigskip` et `\vfill`, entre autres, sont prédéfinis. La commande `\vspace` accepte également comme argument la commande `\stretch{`⟨*poids*⟩`}`.

On accède aux espacements `\medskip` et `\bigskip` par des entrées du menu LaTeX ⇒ Espacements.

C.2.2 Longueurs

Le langage TEX, sur lequel est bâti LATEX, réalise des calculs sur des longueurs pour un grand nombre de tâches. En interne, il travaille sur une unité appelée point d'échelle et notée sp (pour scale point), mais permet de spécifier des longueurs en utilisant plusieurs systèmes d'unités. Le tableau C.1 en dresse la liste complète ainsi que certaines équivalences.

	Unité	Définition	en pt	en in	en mm
sp	scale point	—	$1{,}526 \cdot 10^{-5}$	$2{,}111 \cdot 10^{-7}$	$5{,}363 \cdot 10^{-6}$
pt	point	65536sp	1	0,01384	0,3515
pc	pica	12pt	12	0,1660	4,218
in	pouce	72.27pt	72,27	1	25,4
bp	point *PS*	1in = 72bp	1,00375	0,01389	0,3528
dd	point Didot	1157dd = 1238pt	0,9346	0,01293	0,3285
cc	cicéro	12dd	11,21	0,1552	3,942
cm	centimètre	2.54in = 1cm	28,45	0,3937	10
mm	millimètre	0.1cm	2,845	0,03937	1
ex	hauteur d'œil	—	—	—	—
em	cadrat	—	—	—	—

Tableau C.1 : **Unités de longueurs accessibles**

C

On remarquera au passage que l'unité de base de TEX mesure environ $5{,}363 \times 10^{-6}$ millimètre, c'est-à-dire 5,363 nanomètres. À titre de comparaison, la longueur d'onde

la plus courte visible par l'œil humain est d'environ 400 nanomètres. La précision de TEX ne risque donc pas d'être dépassée par la définition d'un périphérique d'impression... même prévu pour graver des microfilms !

En pratique, les unités les plus couramment employées sont le point, le millimètre et le centimètre, ainsi que les deux unités dépendant de la fonte en cours (ex et em). Pour ces deux dernières, il n'y a pas de règle obligatoire : c'est le concepteur de la fonte qui fixe leurs valeurs. Assez souvent, `1em` correspond au corps de fonte : avec une fonte 10 points, cette unité vaut 10 points. De même, la largeur des chiffres d'une fonte vaut fréquemment `0.5em`. Pour l'unité ex, il s'agit de la hauteur des minuscules sans jambage ni hampe. Là aussi, ce n'est qu'une indication et non une règle précise.

Il est également possible de définir des longueurs un peu comme on définit des commandes. Le nom d'une longueur suit les mêmes règles qu'un nom de macro avec des lettres. La commande qui permet de créer une longueur est `\newlength`.

```
\newlength{\⟨nom⟩}
```

Une fois cette longueur définie, on peut lui donner une valeur ou effectuer des calculs.

```
\setlength{\⟨nom⟩}{longueur}
\settowidth{\⟨nom⟩}{matériel}
\settoheight{\⟨nom⟩}{matériel}
\settodepth{\⟨nom⟩}{matériel}
\addtolength{\⟨nom⟩}{longueur}
```

La commande `\setlength` affecte la ⟨*longueur*⟩ à \⟨*nom*⟩. Les trois commandes suivantes servent à donner à \⟨*nom*⟩ respectivement la largeur, la hauteur et la profondeur du ⟨*matériel*⟩. Enfin, `\addtolength` ajoute la ⟨*longueur*⟩ à celle déjà contenue dans \⟨*nom*⟩. Lorsqu'on passe une ⟨*longueur*⟩ en tant que paramètre, il est possible de l'indiquer de façon directe avec un nombre décimal (éventuellement négatif), suivi d'une unité. On peut également utiliser un autre \⟨*nom*⟩ de longueur pour cela, et même utiliser la syntaxe ⟨*coeff*⟩\⟨*nom*⟩ où ⟨*coeff*⟩ est un nombre décimal (éventuellement négatif) pour obtenir le produit d'une dimension par un nombre. Des exemples sont donnés ci-dessous.

Avec LATEX, les longueurs peuvent être des *ressorts* (on rencontre également le terme *longueur élastique*). Il s'agit d'un concept très puissant et très souple. On a vu que pour indiquer la valeur d'une longueur, on écrivait un nombre décimal suivi d'une unité de longueur. En réalité, on peut aussi ajouter une composante `plus` et une composante `minus`. Par exemple :

```
\setlength{\largeur}{3mm plus 1mm minus 0.5mm}
```

Lorsqu'une longueur est spécifiée de cette façon, elle devient un ressort. La partie `3mm` est la longueur naturelle du ressort, la partie `plus` indique son étirement possible et la partie `minus` sa compression possible. En d'autres termes, avec

```
\hspace{\largeur}
```

où \largeur est définie comme précédemment, l'espacement horizontal produit est normalement de 3 millimètres mais peut en fait atteindre 4 millimètres (3+1) ou bien diminuer jusqu'à 2,5 millimètres (3 – 0,5) en fonction des circonstances, par exemple pour mieux remplir une ligne de texte.

Les composantes d'étirement et de compression peuvent utiliser des unités spéciales pour indiquer l'infini : `fil`, `fill`, et `filll`. Il s'agit d'infinis de plus en plus « grands ». Par exemple, `0.1fill` est infiniment plus grand que `10000fil`. En pratique, `fil` sert dans certaines commandes internes et il vaut mieux ne pas l'utiliser lorsqu'on ne sait pas ce que l'on fait, tandis que `filll` est d'un emploi assez rare.

Prenons l'exemple de la longueur prédéfinie \parskip. Il s'agit d'un ressort placé automatiquement entre deux paragraphes et qui vaut `0pt plus 1pt` par défaut. Cela signifie donc que l'espace entre deux paragraphes est normalement nul mais qu'il peut atteindre 1 point afin de mieux remplir une page.

La commande prédéfinie \stretch est plus surprenante. Elle demande un paramètre et \stretch{⟨*n*⟩} est équivalente à `0pt plus` ⟨*n*⟩`fill`. C'est-à-dire que la longueur naturelle est nulle mais que l'étirement est infini.

C-8

```
\setlength{\parindent}{0pt}
G\hspace{\stretch{1}}D\par
G\hspace{\stretch{1}}M%
\hspace{\stretch{1}}D\par
G\hspace{\stretch{2}}X%
\hspace{\stretch{3}}D
```

```
G                    D
G          M         D
G      X             D
```

Dans cet exemple, le « M » est placé au milieu de la ligne, tandis que le « X » est placé aux deux cinquièmes de la ligne.

Cet exemple modifie la longueur prédéfinie \parindent qui indique le *retrait* de chaque paragraphe. LaTeX propose d'autres espacements prédéfinis. Ils sont répertoriées dans le tableau C.2.

Longueur	Signification
Espacements horizontaux	
\!	espace fine négative (mathématiques)
\,	espace fine (un sixième de *cadratin*)
\:	espace moyenne (mathématiques)
\;	grande espace (mathématiques)
\enspace	demi-*cadratin* (`0.5em`)
\quad	*cadratin* (`1em`)
\qquad	double *cadratin* (`2em`)
\hfill	équivalent à `\hspace{\stretch{1}}`
Espacements verticaux	
\smallskip	un quart de l'interlignage
\medskip	la moitié de l'interlignage
\bigskip	un interlignage
\vfill	équivalent à `\vspace{\stretch{1}}`

Tableau C.2 : **Espacements prédéfinis dans LaTeX**

La commande \hfill place un blanc qui prendra toute la place horizontale disponible. Les commandes \hrulefill et \dotfill font exactement la même chose, mais, au lieu d'un blanc, la première tracera un trait et la deuxième remplira l'espace par des points régulièrement espacés. C'est la deuxième commande qui a été utilisée pour composer les points de suite qui apparaissent dans les sommaires de cet ouvrage.

On fera attention à ce que la commande \hfill n'a aucune action en début de ligne ou en fin de ligne. Pour cela, on dispose de la variante étoilée de \hspace. Les deux autres commandes n'ont pas cette propriété comme le montre l'exemple ci-dessous pour \hrulefill :

C-9

```
\noindent Nom : \dotfill{}
Prénom : \dotfill{}
\hspace*{\stretch{1}}
\par\noindent\hrulefill
```

Nom : ..... Prénom : .....

Ici, les largeurs des deux séries de points de suite se partagent équitablement l'espace disponible sur la ligne avec la commande \hspace*.

C.2.3 Compteurs

Une autre notion importante sous LaTeX est celle de compteur. Les nombres qui apparaissent dans certaines constructions sont en fait gérés par des compteurs. Ainsi en est-il des numéros de chapitres ou de sections, des formules mathématiques numérotées (cf. chapitre 4), de la pagination, etc.

La commande \newcounter permet de créer un compteur. Comme pour les longueurs, un compteur est accessible par un nom. En revanche, les noms des compteurs ne commencent pas par une *contre-oblique*.

```
\newcounter{⟨compteur⟩}[⟨cpt-dep⟩]
```

Cette commande crée le ⟨*compteur*⟩. L'argument optionnel ⟨*cpt-dep*⟩ indique un compteur qui prendra le ⟨*compteur*⟩ défini sous sa dépendance. On verra ce que cela signifie un peu plus loin. On peut noter dès à présent la place inhabituelle de cet argument optionnel.

Lorsqu'un nouveau compteur est créé, il est automatiquement initialisé avec une valeur nulle. Pour affecter une autre valeur à un compteur, on peut utiliser les commandes \setcounter ou \addtocounter.

```
\setcounter{⟨compteur⟩}{⟨valeur⟩}
\addtocounter{⟨compteur⟩}{⟨valeur⟩}
```

La première affecte la ⟨*valeur*⟩ au ⟨*compteur*⟩, tandis que la deuxième ajoute la ⟨*valeur*⟩ au contenu du ⟨*compteur*⟩. Les compteurs ne gèrent que les nombres entiers (y compris négatifs), ils ne sont absolument pas prévus pour les nombres décimaux. Comme pour tout ce qui concerne les nombres entiers, l'intervalle de validité de la ⟨*valeur*⟩ est $[-2147483647, 2147483647]$ ($[-(2^{31}-1), 2^{31}-1]$ et non pas $[-2147483647, 2147483648]$ comme dans la plupart des langages de programmation, Knuth a absolument voulu que chaque nombre ait son opposé).

C

Une fois qu'un compteur est défini et qu'une valeur lui a été affectée, il va falloir afficher son contenu ou l'utiliser dans un contexte qui réclame un nombre entier. Pour cela, LaTeX ne propose pas moins de sept commandes.

```
\value{⟨compteur⟩}   \arabic{⟨compteur⟩}   \fnsymbol{⟨compteur⟩}
\roman{⟨compteur⟩}   \Roman{⟨compteur⟩}
\alph{⟨compteur⟩}    \Alph{⟨compteur⟩}
```

La commande \value permet d'utiliser le ⟨*compteur*⟩ dans les contextes où LaTeX demande un nombre entier. Les six autres commandes permettent d'afficher le contenu du ⟨*compteur*⟩ respectivement en écriture dite arabe, avec une liste de symboles traditionnels dans les appels de notes des documents anglo-saxons, en minuscules *romaines*, en majuscules *romaines*, en lettres minuscules et en lettres majuscules.

C-10

```
\newcounter{increment}\newcounter{num}
\setcounter{increment}{2}
\newcommand\prochain[1]{%
  \addtocounter{num}{\value{increment}}%
  #1{num}\quad
}
\prochain{\arabic}\prochain{\roman}%
\prochain{\Roman}\prochain{\fnsymbol}%
\prochain{\alph}\prochain{\Alph}%
```

2 iv VI †† j L

L'augmentation de la valeur d'un compteur d'une unité est tellement utile que LaTeX propose deux raccourcis pour l'effectuer.

```
\stepcounter{⟨compteur⟩}   \refstepcounter{⟨compteur⟩}
```

Ces deux commandes mettent à zéro les compteurs dépendant du compteur qui vient d'être incrémenté (celui spécifié dans l'argument optionnel de \newcounter). Par exemple, le compteur subsection dépend du compteur section. Lorsqu'on utilise la macro \section, celle-ci exécute \refstepcounter{section}. On a alors automatiquement une remise à zéro du compteur subsection.

D'autre part, la commande \refstepcounter permet aux prochaines commandes \label de se référer au compteur venant d'être incrémenté. Dans le cas de \section, cela permet à une commande \label qui suit de se référer au numéro de cette section.

LaTeX propose un mécanisme permettant d'afficher un compteur avec une forme par défaut. Chaque fois qu'un compteur est créé avec \newcounter{⟨*cpt*⟩}, LaTeX crée également la commande \the⟨*cpt*⟩ avec la définition \arabic{⟨*cpt*⟩}.

On est libre de redéfinir ces commandes comme on le souhaite. Par exemple,

```
\renewcommand\theexercice{\thesection.\arabic{exercice}}
```

permet d'afficher le compteur exercice, créé au préalable, sous la forme du numéro de section suivi d'un point, suivi du numéro d'exercice en notation arabe.

C

C.2.4 Package calc

LaTeX permet quelques calculs de base sur les longueurs et les compteurs mais cela reste très limité. Pour les longueurs, on peut ajouter ou multiplier par une constante et, pour les compteurs, on ne peut qu'ajouter. De plus, on ne peut réaliser qu'une seule opération à la fois.

Par exemple, si `\lgA`, `\lgB` et `\lgC` sont trois longueurs, pour que `\lgA` soit égale à la somme des deux autres, il faut procéder en deux étapes :

```
\setlength{\lgA}{\lgB}
\addtolength{\lgA}{\lgC}
```

Pour des calculs plus complexes, ces décompositions deviennent vraiment pénibles. Le package calc offre une syntaxe bien plus agréable pour réaliser de tels calculs.

Ce package redéfinit les commandes d'affectation et d'addition sur les longueurs et les compteurs, donc les quatre macros : `\setlength`, `\setcounter`, `\addtolength` et `\addtocounter`, afin que leur deuxième argument puisse être une expression sous forme infixe (celle que tout le monde utilise depuis l'école primaire) au lieu d'une longueur ou d'un nombre seul.

On a accès aux quatre opérations arithmétiques avec leurs symboles usuels en informatique, à savoir `+`, `-`, `*` et `/`. Les règles de priorité sont respectées et il est possible d'utiliser des parenthèses.

N'importe quelle expression est acceptée aux conditions suivantes :

- les additions et soustractions ne peuvent mélanger longueurs et nombres ;
- dans une expression donnant une longueur, la première opérande qui apparaît doit être une longueur ;
- pour multiplier par un nombre décimal, il faut utiliser la commande `\real` (strictement parlant ce n'est pas tout à fait vrai mais autant prendre de bonnes habitudes dès le départ) ;
- pour diviser deux longueurs, il faut utiliser la commande `\ratio` ; comme on est en droit de s'y attendre, la division de deux longueurs est un nombre, pas une dimension (cela signifie que la partie décimale du résultat sera tronquée) ;
- si le résultat doit être la valeur d'un compteur, il est possible d'utiliser des nombres décimaux mais la partie décimale du résultat sera tronquée.

La première règle est évidente (`2cm+4` n'a pas de signification claire). La deuxième règle est plus surprenante. Elle indique, par exemple, que `4*2cm` est illégal et qu'il faut l'écrire sous la forme `2cm*4`.

Avec cette syntaxe, l'affectation présentée au début de cette section peut maintenant s'écrire plus simplement :

```
\setlength{\lgA}{\lgB+\lgC}
```

Le package calc offre également trois commandes permettant d'accéder aux dimensions d'un matériel quelconque à l'intérieur d'un calcul.

`\widthof{`⟨*matériel*⟩`}` `\heightof{`⟨*matériel*⟩`}` `\depthof{`⟨*matériel*⟩`}`

Ces commandes fournissent respectivement la largeur, la hauteur et la profondeur du ⟨*matériel*⟩. La hauteur est la longueur située au-dessus de la ligne de base et la profondeur celle située au-dessous.

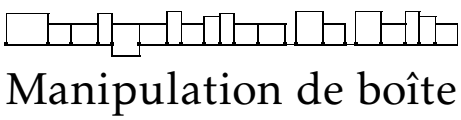
C

C.3 Manipulation de boîtes

La notion de boîte est tout à fait fondamentale. En réalité, lorsque LaTeX compose du texte, il ne fait que manipuler des boîtes : un caractère est vu comme une boîte (le dessin du caractère n'intervient pas à ce stade), une ligne de texte est une boîte où sont juxtaposées les boîtes de caractères, un paragraphe est une boîte où sont empilées les boîtes des lignes de texte et une page est une boîte où sont empilés des paragraphes.

Une boîte est donc un objet rectangulaire qui a une hauteur, une profondeur et une largeur, et dont le contenu peut être arbitrairement complexe. Une fois que du matériel est « en boîte », il est manipulé comme un tout : on peut tout à fait considérer qu'il s'agit d'une sorte de caractère (éventuellement énorme). Une boîte a également un point de référence. Il s'agit d'un point situé au niveau de sa frontière gauche au niveau de la ligne de base. Les boîtes sont alignées verticalement avec leurs points de référence placés au niveau de la ligne de base.

Prenons le texte « Manipulation de boîte ». Du point de vue de LaTeX, il s'agit d'un amoncellement de boîtes qui pourrait être décrit de la façon suivante (on a récrit le texte au-dessous pour bien voir la correspondance et on a représenté le point de référence de chaque boîte par un point noir) :



Manipulation de boîte

L'utilisateur a une grande liberté pour gérer certaines boîtes. Par exemple, le curieux logo LaTeX est obtenu en décalant verticalement et horizontalement des boîtes de caractères.

Les boîtes qui peuvent être manipulées directement par l'utilisateur sont de trois types : les boîtes horizontales (dites boîtes LR pour Left-Right, c'est-à-dire avec une écriture de gauche à droite), les boîtes verticales (dites boîtes de paragraphes, car elles permettent de composer du texte sur plusieurs lignes), et enfin les boîtes de réglures qui produisent des rectangles pleins.

C.3.1 Boîtes horizontales

Il existe deux macros qui permettent de produire une boîte horizontale simple : \mbox et \makebox.

```
\mbox{⟨matériel⟩}
\makebox[⟨largeur⟩][⟨position⟩]{⟨matériel⟩}
```

La commande \makebox place le ⟨*matériel*⟩ dans une boîte horizontale dont on peut préciser la ⟨*largeur*⟩ et la façon d'aligner ce matériel avec l'argument ⟨*position*⟩. Si ces arguments optionnels ne sont pas précisés, la boîte a la largeur du matériel. La commande \mbox n'est qu'un raccourci de \makebox sans aucun argument optionnel.

L'argument ⟨*position*⟩ peut prendre quatre valeurs : `c` (valeur par défaut), `r`, `l` et `s`, indiquant respectivement un alignement centré, à droite, à gauche, et s'étendant dans toute la boîte (`s` pour *stretch*, *étirement*). Pour le dernier cas, il devra exister des espacements élastiques pour permettre cet étalement.

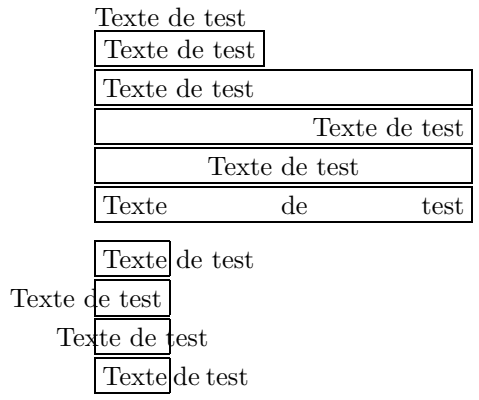
Lorsque la largeur indiquée est inférieure à celle du matériel, il y aura un débordement. On le verra à l'exemple C-11.

Il existe deux commandes voisines de `\mbox` et `\makebox` qui permettent d'encadrer la boîte. Il s'agit de `\fbox` et `\framebox`.

```
\fbox{⟨matériel⟩}
\framebox[⟨largeur⟩][⟨position⟩]{⟨matériel⟩}
```

C-11

```
Texte de test\par
\fbox{Texte de test}\par
\framebox[5cm][l]{Texte de test}\par
\framebox[5cm][r]{Texte de test}\par
\framebox[5cm][c]{Texte de test}\par
\framebox[5cm][s]{Texte de test}\par
\vspace{2mm}
\framebox[1cm][l]{Texte de test}\par
\framebox[1cm][r]{Texte de test}\par
\framebox[1cm][c]{Texte de test}\par
\framebox[1cm][s]{Texte de test}\par
```



L'exemple montre le comportement des deux options de la commande `\framebox`. Lorsque la largeur indiquée est plus grande que le texte, les options `l`, `r` et `c` se comportent comme prévu. En ce qui concerne l'option `s`, c'est un peu plus compliqué. Le texte contient deux espaces et il s'agit bien d'éléments qui peuvent s'agrandir ou se rétrécir. En revanche, LaTeX ne permet normalement pas aux espaces de s'agrandir au-delà d'une certaine largeur, qui est très largement dépassée dans l'exemple. Les ressorts ont en fait le droit de s'étendre plus que ce qu'ils sont capables de faire mais il y a un rappel à l'ordre au niveau du rapport de compilation à propos d'une `Underfull \hbox` (boîte pas assez pleine).

Lorsqu'on spécifie une largeur plus petite que le ⟨*matériel*⟩, le texte déborde de la boîte. Ainsi, la boîte avec `[1cm][l]` a son contenu qui déborde à droite, et les autres débordent à gauche ou des deux côtés. La boîte avec les options `[1cm][s]` déborde également à droite, mais, ici, les ressorts contenus dans le ⟨*matériel*⟩ (c'est-à-dire les deux espaces) ont été comprimés au maximum. En regardant attentivement la dernière ligne de l'exemple, on peut s'apercevoir que le texte est un peu moins large que dans les autres cas. Si les ressorts peuvent s'étendre plus que ce qui est permis, il n'en est pas de même au niveau de la compression (heureusement, sinon tous les mots se chevaucheraient).

Le cadre qui entoure le ⟨*matériel*⟩ peut être ajusté avec les deux longueurs prédéfinies `\fboxrule`, qui indique l'épaisseur des lignes du cadre (`0.4pt` par défaut), et `\fboxsep`, qui indique la distance entre le ⟨*matériel*⟩ et le cadre (`3pt` par défaut).

C

C-12

```
Une \fbox{boîte}\par
\setlength{\fboxrule}{1.5pt}
\setlength{\fboxsep}{6pt}
Une \framebox[5cm][c]{autre boîte}
```

Une boîte
Une autre boîte

Les commandes qui permettent de spécifier la largeur de la boîte peuvent utiliser la macro \width, qui indique alors la largeur naturelle du ⟨*matériel*⟩. On a également accès aux commandes \height, \depth et \totalheight, qui donnent respectivement la hauteur, la profondeur et la somme de la hauteur et de la profondeur du ⟨*matériel*⟩. Ces commandes permettent, par exemple, de modifier les dimensions d'une boîte en fonction de ses dimensions naturelles sans avoir à faire de calculs compliqués dessus à l'aide de longueurs auxiliaires.

La commande \raisebox permet de décaler verticalement une boîte horizontale.

```
\raisebox{⟨décalage⟩}[⟨hauteur⟩][⟨profondeur⟩]{⟨matériel⟩}
```

Le ⟨*matériel*⟩ est composé en tant que boîte horizontale puis surélevé du ⟨*décalage*⟩. Si cette longueur est négative, on obtient un abaissement de la boîte. Les paramètres optionnels ⟨*hauteur*⟩ et ⟨*profondeur*⟩ permettent de fixer les dimensions verticales de la boîte sans tenir compte du contenu.

C.3.2 Boîtes verticales

Les boîtes verticales peuvent composer leur matériel sur plusieurs lignes, voire plusieurs paragraphes. Pour cela, il faudra toujours indiquer la largeur. Une commande et un environnement permettent d'obtenir de telles boîtes.

```
\parbox[⟨pos⟩][⟨hauteur⟩][⟨int⟩]{⟨largeur⟩}{⟨matériel⟩}
\begin{minipage}[⟨pos⟩][⟨hauteur⟩][⟨int⟩]{⟨largeur⟩}
  ⟨matériel⟩
\end{minipage}
```

Cette commande et cet environnement placent le ⟨*matériel*⟩ dans une boîte verticale de ⟨*largeur*⟩ spécifiée. Le premier argument optionnel indique la position de la boîte par rapport à la ligne de base : c pour centré (option par défaut), t pour un alignement sur la ligne supérieure, et b pour un alignement sur la ligne inférieure de la boîte. On peut également indiquer une ⟨*hauteur*⟩. Si celle-ci n'est pas précisée, c'est la hauteur du ⟨*matériel*⟩ qui sera utilisée. Enfin, si une ⟨*hauteur*⟩ est indiquée, on peut spécifier la façon d'aligner verticalement le ⟨*matériel*⟩ dans la boîte. On retrouve pour cela les trois valeurs possibles de l'option ⟨*pos*⟩ avec la même signification ainsi que la valeur s (stretch) pour que le ⟨*matériel*⟩ occupe toute la hauteur indiquée. Dans ce cas, comme pour ce qui se passait avec les boîtes horizontales, il faut qu'il y ait des ressorts verticaux dans le ⟨*matériel*⟩.

Comme son nom l'indique, l'environnement minipage se comporte comme une sorte de page. Par exemple, contrairement à la commande \parbox, il est possible d'y placer des notes de bas de page, un texte en plusieurs colonnes, des tableaux, etc.

C

C-13
```
\usepackage{calc}
Ligne de base
\fbox{%
  \begin{minipage}[b][\height+15pt][s]{1in}
    Premier paragraphe dans la boîte.
    \par\vfill
    Deuxième paragraphe.
  \end{minipage}}
```

Ligne de base | Premier paragraphe dans la boîte. Deuxième paragraphe.

Dans cet exemple, on a composé deux paragraphes dans une minipage, le tout étant passé en argument de \fbox. On peut remarquer l'utilisation de \height pour obtenir la hauteur naturelle du contenu de la minipage, ainsi que l'utilisation du package calc pour effectuer facilement un calcul permettant d'augmenter cette hauteur de 15 points. Comme la hauteur devient plus grande que la hauteur naturelle et qu'on a spécifié une option [s], il est important de placer des ressorts verticaux suffisants. Ici, un \vfill suffit amplement.

C.3.3 Boîtes de réglures

Les boîtes de réglures sont des boîtes rectangulaires pleines. Par défaut, elles apparaîtront noires mais il est possible de modifier leur couleur avec le package xcolor (cf. chapitres 3 et 8).

La commande \rule permet d'obtenir de telles boîtes.

```
\rule[⟨décalage⟩]{⟨largeur⟩}{⟨hauteur⟩}
```

Cette commande trace un rectangle de ⟨*largeur*⟩ et ⟨*hauteur*⟩ spécifiées. Par défaut, ce rectangle est posé sur la ligne de base mais il est possible de le surélever avec l'argument optionnel ⟨*décalage*⟩. Si cette longueur est négative, le rectangle sera abaissé.

C-14
```
\rule[0.5ex]{5mm}{0.4pt}
ligne de texte
\rule[0.5ex]{5mm}{0.4pt}
```

— ligne de texte —

Dans l'exemple, on a utilisé l'unité ex afin d'obtenir des traits positionnés verticalement au milieu des lettres minuscules.

Il est possible de donner une ⟨*largeur*⟩ ou une ⟨*hauteur*⟩ nulles. On obtient alors une boîte invisible mais qui occupera cependant une certaine hauteur ou une certaine largeur. En pratique, on n'utilise pas de boîtes de réglures de hauteur nulle car il existe d'autres façons de procéder : commande \hspace et boîte horizontale de largeur imposée. En revanche, les réglures de largeur nulle sont très utiles.

C-15
```
\newcommand\Fbox[1]{%
  \fbox{\rule[-3.6pt]{0pt}{12pt}#1}%
}
\fbox{que} \fbox{du} \fbox{texte}\par
\Fbox{que} \Fbox{du} \Fbox{texte}
```

que du texte
que du texte

C

Dans l'exemple, lorsqu'on se sert de la commande \Fbox au lieu de \fbox, on s'assure que tous les cadres auront la même profondeur et la même hauteur sans perturber la largeur. En réalité, les valeurs employées pour la commande \rule ne sont pas du tout quelconques. Lorsqu'on travaille avec une fonte en corps 10 points, la hauteur de 12 points correspond à l'interlignage habituel, et la profondeur de 3,6 points correspond à 0,3 fois la hauteur totale. Si l'on ne veut pas être dépendant du corps de la fonte principale, la valeur de l'interlignage est mémorisée dans la longueur \baselineskip. Une commande plus générale serait donc :

```
\newcommand\Fbox[1]{%
  \fbox{\rule[-0.3\baselineskip]{0pt}{\baselineskip}}
}
```

En fait, cette commande \rule avec les valeurs utilisées ci-dessus est tellement utile qu'elle existe sous forme de raccourci : \strut. Elle permet d'être certain que le matériel occupera la place verticale d'une ligne de texte normale.

C.3.4 Sauvegarde de boîtes

Une boîte peut être sauvegardée pour être utilisée plus loin dans le document. Par exemple, pour en calculer ses dimensions, ou, comme nous allons le voir dans l'exemple C-16 pour sauvegarder le corps d'un environnement afin de travailler dessus par la suite. Pour cela, il faut tout d'abord déclarer une boîte en lui donnant un nom à l'aide de la commande \newsavebox.

```
\newsavebox{\⟨boîte⟩}
```

Une fois la \⟨*boîte*⟩ déclarée, on peut sauvegarder un texte dans cette boîte avec les commandes \sbox et \savebox ou l'environnement lrbox.

```
\sbox{\⟨boîte⟩}{⟨matériel⟩}
\savebox{\⟨boîte⟩}[⟨largeur⟩][⟨position⟩]{⟨matériel⟩}
\begin{lrbox}{\⟨boîte⟩}
  ⟨matériel⟩
\end{lrbox}
```

Chaque fois, on sauvegarde le ⟨*matériel*⟩ dans la \⟨*boîte*⟩. Les commandes \sbox et \savebox sont les équivalents des commandes \mbox et \makebox vues page 384. L'environnement lrbox a exactement le même rôle que la commande \sbox mais peut être utilisé dans une définition d'environnement afin de sauvegarder le corps dans une boîte et d'effectuer un certain travail dessus.

Les boîtes que l'on sauvegarde de cette façon sont des boîtes horizontales. Si l'on désire sauvegarder du matériel vertical, il suffit d'emboîter une commande \parbox ou un environnement minipage.

C-16

```
\newsavebox{\ctbox}
\newenvironment*{cadretitre}[2]
{%
  \begin{lrbox}{\ctbox}
  \begin{minipage}{#1}
    {\centering \textbf{#2}\par}
    \hrulefill\par\itshape
}
{%
  \end{minipage}\end{lrbox}
  \fbox{\usebox{\ctbox}}
}
\begin{cadretitre}{5cm}{Phage T4}
  Après un rappel sur la réplication
  du phage T4, nous étudierons plus
  spécialement le crossing-over.
\end{cadretitre}
```

Phage T4

Après un rappel sur la réplication du phage T4, nous étudierons plus spécialement le crossing-over.

C.4 Programmation structurée

Un langage de programmation complet se doit de disposer de structures conditionnelles et de pouvoir réaliser des boucles. LaTeX propose quelques commandes pour cela, mais il s'agit de macros internes dont la syntaxe est assez déroutante. Plusieurs packages permettent de disposer de telles structures en proposant une syntaxe plus agréable et plus puissante.

C.4.1 Structures conditionnelles

Le package ifthen offre la commande `\ifthenelse` pour les actions conditionnelles.

`\ifthenelse{`⟨*condition*⟩`}{`⟨*vrai*⟩`}{`⟨*faux*⟩`}`

Cette commande évalue la ⟨*condition*⟩. Si celle-ci est vraie, le code ⟨*vrai*⟩ est exécuté, sinon, c'est le code ⟨*faux*⟩ qui l'est. Les codes ⟨*vrai*⟩ et ⟨*faux*⟩ sont totalement libres (à certaines obligations près comme celle d'équilibrer les accolades). Il reste maintenant à étudier la façon d'écrire les conditions.

La première commande que nous allons voir sert à comparer deux chaînes.

`\equal{`⟨*chaîne*$_1$⟩`}{`⟨*chaîne*$_2$⟩`}`

La condition sera vraie si les deux ⟨*chaîne*$_i$⟩ sont identiques.

C

L'exemple C-1 indiquait une façon de concevoir une commande permettant de composer les siècles. La technique employée n'était pas optimale car le paramètre optionnel était théoriquement connu une fois la valeur du paramètre obligatoire donnée. Il fallait employer \ier lorsqu'on parlait du premier siècle et \ieme dans tous les autres cas. Une façon bien plus agréable et plus sûre consiste à utiliser des tests.

C-17

```
\usepackage{ifthen}
\newcommand*\siecle[1]{\textsc{#1}%
  \ifthenelse{\equal{#1}{i}}
             {\ier}{\ieme}}
Du \siecle{i} au \siecle{iii} siècle,
```

Du Ier au IIIe siècle,

On peut également comparer deux nombres avec un des trois opérateurs =, < et >.

C-18

```
\usepackage{ifthen}
Nous sommes \ifthenelse{\the\time<721}
            {le matin}{l'après-midi}
```

Nous sommes l'après-midi

Cet exemple utilise l'instruction \the\time qui donne le nombre de minutes écoulées depuis minuit. En l'occurrence, la valeur 720 correspond donc à midi.

La commande \isodd demande un nombre en argument et retourne la valeur vraie si ce nombre est impair. Une utilisation évidente de cette commande est d'élaborer un test sur la parité de la page en cours. Malheureusement le compteur page qui est censé contenir le numéro de page est victime du mode asynchrone de fonctionnement de LaTeX et il ne contient pas toujours le vrai numéro de page. Une façon d'obtenir ce numéro est d'utiliser le mécanisme des étiquettes \label et \pageref. La commande \isodd a été spécialement conçue pour accepter une syntaxe du type :

```
\label{ici}Nous sommes en page
\ifthenelse{\isodd{\pageref{ici}}}{paire}{impaire}
```

Pour comparer des longueurs, il faudra utiliser la commande \lengthtest avec les mêmes opérateurs que pour les compteurs.

C-19

```
\usepackage{ifthen}
\newlength{\largA}\newlength{\largB}
\newcommand\ChainesMemeLongueur[2]{\par
  \settowidth{\largA}{#1}%
  \settowidth{\largB}{#2}%
  \ifthenelse{\lengthtest{\largA>\largB}}
    {#1\par\makebox[\largA][s]{#2}\par}
    {\makebox[\largB][s]{#1}\par#2\par}}
\ChainesMemeLongueur
  {voici deux chaînes}
  {qui ont la même largeur}
```

voici deux chaînes
qui ont la même largeur

Toutes les conditions élémentaires que nous avons vues peuvent être assemblées pour former des conditions complexes. On dispose pour cela des connecteurs logiques \and, \or et \not, ainsi que des parenthèses \(et \).

C

C.4.2 Boucles

Le package ifthen propose la commande \whiledo pour réaliser des boucles while.

```
\whiledo{⟨condition⟩}{⟨corps⟩}
```

Le ⟨*corps*⟩ sera exécuté tant que la ⟨*condition*⟩ sera vraie. Cette ⟨*condition*⟩ sera écrite comme celle de la commande \ifthenelse.

C-20

```
\usepackage{ifthen}
\newcounter{repet}
\newcommand\punition[2]{%
 \setcounter{repet}{#1}%
 \whiledo{\value{repet}>0}{%
  #2\par\addtocounter{repet}{-1}}
}

\punition{5}{Je dois éteindre mon
  portable en cours.}
```

Je dois éteindre mon portable en cours.
Je dois éteindre mon portable en cours.
Je dois éteindre mon portable en cours.
Je dois éteindre mon portable en cours.
Je dois éteindre mon portable en cours.

Les boucles while sont très utiles mais, dans l'exemple précédent, un programmeur aurait plutôt choisi une boucle for. Le package multido offre cette possibilité grâce à sa puissante commande \multido.

```
\multido{⟨initialisation⟩}{⟨repet⟩}{⟨corps⟩}
```

Le ⟨*corps*⟩ va être répété ⟨*repet*⟩ fois. La partie ⟨*initialisation*⟩ indique la façon d'initialiser la ou les variables de boucle ainsi que la façon de les incrémenter à chaque tour. Une ⟨*initialisation*⟩ est une liste d'initialisations élémentaires séparées par des virgules. Chaque initialisation élémentaire est de la forme :

```
⟨var⟩ = ⟨init⟩ + ⟨incrément⟩
```

Au début de la boucle, la variable ⟨*var*⟩ est initialisée à ⟨*init*⟩, puis, chaque fois qu'un tour de boucle se termine, cette variable est incrémentée de la valeur ⟨*incrément*⟩.

L'⟨*initialisation*⟩ peut être vide. Dans ce cas, la commande permet de répéter simplement un texte. Voici comment l'exemple précédent peut être écrit avec cette structure de boucle :

C-21

```
\usepackage{multido}
\newcommand\punition[2]{%
 \multido{}{#1}{#2\endgraf}
}

\punition{5}{Je dois éteindre mon
  portable en cours.}
```

Je dois éteindre mon portable en cours.
Je dois éteindre mon portable en cours.
Je dois éteindre mon portable en cours.
Je dois éteindre mon portable en cours.
Je dois éteindre mon portable en cours.

C

Reconnaissons que c'est plus simple ! On a été cependant obligé d'employer la commande \endgraf. Il s'agit d'un équivalent de \par mais qui est permis dans les arguments de commandes n'acceptant normalement pas plusieurs paragraphes.

Lorsqu'on travaille avec des initialisations, la première lettre de la commande servant de variable indique son type :

d ou **D.** La valeur initiale et l'incrément sont des dimensions (longueurs).

i ou **I.** La valeur initiale et l'incrément sont des nombres entiers.

n ou **N.** La valeur initiale est un nombre à virgule et l'incrément devra obligatoirement avoir le même nombre de chiffres après la virgule.

r ou **R.** La valeur initiale et l'incrément sont des nombres à virgule.

C-22

```
\usepackage{multido}
\multido{\in=2+1,
         \npi=6.28318530+3.14159265}
        {4}{$\in\pi \approx \npi$\endgraf}
```

$2\pi \approx 6.28318530$
$3\pi \approx 9.42477795$
$4\pi \approx 12.56637060$
$5\pi \approx 15.70796325$

Rien n'empêche d'emboîter des commandes \multido. D'autre part, les variables de boucle sont protégées. C'est-à-dire que l'on peut utiliser des variables qui sont en réalité des commandes qui existent déjà. C'était le cas dans l'exemple avec \in.

C

Exercices

Exercice C-1 Page de titre

La commande `\maketitle` compose un titre. Pour être plus libre, on peut utiliser l'environnement `titlepage`. Cette liberté se paie par l'obligation de placer soi-même les différents éléments de la page de titre. En particulier, on est souvent amené à positionner des éléments à des emplacements précis. Comment obtenir la page de titre présentée ci-dessous ?

C-23

Université de Munich

Année 1884 Thèse

Théodore Escherich
Le phage T4

Imprimé le 24 septembre 2010

Les textes doivent s'étendre sur toute la hauteur de page, le *retrait* est supprimé, et on réserve deux centimètres à droite du mot « Thèse » (pour son numéro).

En prenant comme base l'espace vertical entre les deux premières lignes, l'espace vertical entre la deuxième ligne et le nom est deux fois plus grand, et l'espace vertical entre le titre et la dernière ligne est trois fois plus grand. Enfin, le nom et le titre sont écrits en gras avec une fonte plus grande que la normale.

Exercice C-2 Arguments optionnels multiples

Une commande comme \parbox utilise plusieurs arguments optionnels. Or LaTeX ne permet de définir que des commandes ayant un seul argument optionnel. La véritable définition de \parbox utilise des commandes LaTeX et TeX de bas niveau pour gérer ce type d'arguments, ce qui dépasse le cadre de cet ouvrage.

En n'utilisant que ce qui a été vu dans ce chapitre, comment définir une commande \lasse qui compose le texte « Les [deux] (assertions) suivant(e)s sont équivalent(e)s » où le terme entre crochets est un argument obligatoire et où les termes entre parenthèses sont les valeurs par défaut de deux arguments optionnels ? D'autre part, on aimerait que les paramètres soient appelés sous la forme suivante :

```
\lasse{trois}
\lasse{quatre}[propositions]
\lasse{deux}[arbres][]
```

c'est-à-dire avec les deux arguments optionnels *après* l'argument obligatoire. Les trois instructions ci-dessus donnent les expressions respectives :

- les trois assertions suivantes sont équivalentes ;
- les quatres propositions suivantes sont équivalentes ;
- les deux arbres suivants sont équivalents.

Exercice C-3 Limites des compteurs

Certaines commandes d'écriture de nombres imposent des intervalles de validité. Ainsi, \alph et \Alph ne fonctionnent évidemment que sur l'intervalle $[1,26]$ puisqu'on doit obtenir les lettres de l'alphabet [1], et \fnsymbol ne propose que neuf symboles différents. Pour \value et \arabic, les limitations sont celles de TeX lui-même, c'est-à-dire l'intervalle $[-2^{31}+1, 2^{31}-1]$.

Il est facile de comprendre pourquoi TeX refuse de composer un nombre négatif ou nul en *romain*. En revanche, sur une installation classique, le code :

```
\documentclass{article}
\newcounter{essai}
\begin{document}
\setcounter{essai}{2147483647}
\Roman{essai}
\end{document}
```

provoque l'erreur de compilation suivante :

```
! TeX capacity exceeded, sorry [pool size=1170118].
<argument> \c@essai

l.5 \Roman{essai}
```

Sauriez-vous expliquer l'origine de cette erreur ?

C

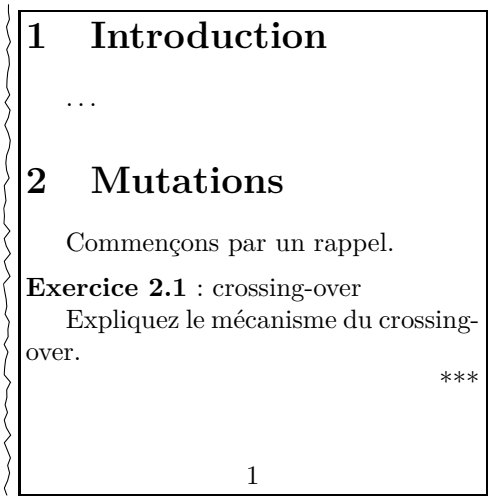
1. Le package alphalph permet de dépasser cette limitation.

Exercice C-4 Exercices numérotés

Dans un document en classe article, on veut définir un environnement exercice qui va permettre de composer des textes d'exercices.

C-24

```
\section{Introduction}
\dots
\section{Mutations}
Commençons par un rappel.
\begin{exercice}{crossing-over}
Expliquez le mécanisme du
crossing-over.
\end{exercice}
```



1 Introduction

...

2 Mutations

Commençons par un rappel.

Exercice 2.1 : crossing-over
Expliquez le mécanisme du crossing-over.
***

1

Cet environnement doit commencer un nouveau paragraphe sans retrait d'alinéa avec un espacement vertical au-dessus, composer le numéro d'exercice en gras suivi d'un titre passé en paramètre. L'énoncé de l'exercice commence un nouveau paragraphe. Après avoir écrit l'énoncé de l'exercice, trois étoiles seront composées en bas à droite pour indiquer la fin de l'exercice et, enfin, l'environnement passera à un nouveau paragraphe en insérant un espacement vertical.

Le numéro d'exercice devra être remis à zéro à chaque début de section et ce numéro sera composé sous la forme ⟨*numéro de section*⟩.⟨*numéro d'exercice*⟩. On pourra également penser aux emplacements où une coupure de page serait malvenue.

Le but de l'exercice est donc de définir cet environnement ainsi que tous les éléments qui pourraient être utiles pour obtenir le résultat montré ci-dessus.

Exercice C-5 Affichage soigné de l'heure

Le but de l'exercice est de concevoir une commande \affH qui prend en paramètre le nombre de minutes écoulées depuis minuit et qui affiche l'heure sous la forme « *h* heures *m* minutes du matin (ou de l'après-midi) ». La commande devra employer le pluriel ou le singulier à bon escient. On pourra aussi traiter les cas particuliers de midi, minuit, et du 0 minute.

C-25

```
\affH{0} --- \affH{720}\par
\affH{1} --- \affH{722}\par
\affH{360}\par
\affH{\the\time}
```

minuit — midi
minuit 1 minute — midi 2 minutes
6 heures du matin
11 heures de l'après-midi

C

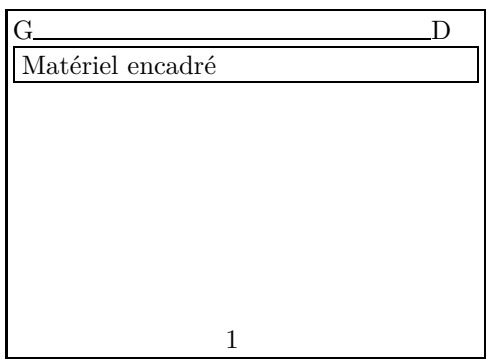
Exercice C-6 Cadre de largeur donnée

En plaçant un environnement minipage en tant qu'argument d'une commande \fbox, on arrive à encadrer facilement un matériel arbitrairement complexe.

Supposons que l'on veuille encadrer du matériel par un cadre faisant exactement la largeur d'empagement. Une approche simpliste conduit à un résultat décevant :

C-26

```
\noindent G\hrulefill D\par
\noindent\fbox{%
  \begin{minipage}{\linewidth}
    Matériel encadré
  \end{minipage}
}
```



Un examen du résultat, même superficiel, montre que le cadre déborde de la page. Pour bien le visualiser sur l'exemple, on a placé une lettre G au niveau de la marge gauche et une lettre D au niveau de la marge droite.

Quelles sont les raisons de ce piteux résultat et comment le corriger ?

Exercice C-7 Surimpression de texte

Comment concevoir une commande \corrige qui compose un texte passé comme premier argument et place par-dessus une suite de caractères identiques. Le caractère devant être répété sera passé comme deuxième argument.

C-27

```
\texttt{Un texte \corrige{raturé}{x} barré}
```

Un texte xxxxxx barré

Dans l'exemple, l'utilisation d'une fonte à chasse fixe et de la lettre « x » permet de simuler une correction comme sur une machine à écrire.

Exercice C-8 Suite de Syracuse

La suite de Syracuse est définie ainsi. On choisit un premier nombre entier strictement positif et on calcule un terme quelconque de la façon suivante : si le terme précédent est pair, on prend sa moitié, sinon on le multiplie par trois et on ajoute un.

Le but de l'exercice est d'écrire une commande \syracuse qui prend la valeur initiale en paramètre et qui affiche la suite des termes jusqu'à arriver à la valeur 1. Par exemple, l'instruction \syracuse{7} donnera le résultat :

C-28

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

C