Troisième partie

Troisième atelier : programmation, macros, compteurs, graphisme, programmes externes

Chapitre 11

Graphisme

11.1 Petit aperçu des possibilités

Au départ[1], ce chapitre n'était là que pour montrer quelques possibilités offertes par LaTeX : il ne s'agissait pas d'étudier ces présentations et les explications étaient minimales.

Mais utiliser LaTeX sans étudier ses possibilités graphiques nous a semblé dangereusement limitatif. En effet, certains objet de LaTeX sont des graphiques même si cela n'apparaît pas immédiatement.

11.1.1 Ce manuel

Les flottants

Le manuel lui-même comprend des constructions qui n'ont pas été évoquées jusqu'ici. Par exemple, les tableaux ayant une légende et un numéro sont ce que LaTeX appelle des " flottants ". Un flottant est une structure qui sera placée dans le document de façon automatique par LaTeX en fonction de la place disponible, donc pas nécessairement au niveau où cette structure a été tapée dans le source. LaTeX prévoit des flottants pour les tableaux et pour les figures. On notera la présence d'une *Liste des tableaux* à la fin du manuel, liste qui a été créée automatiquement.

Inclusion dans un paragraphe

La première version de cet ouvrage présentait des inclusions de dessins dans des paragraphes avec un découpage du texte autour du dessin. Là aussi, tout est fait de façon automatique, le source indique seulement qu'il faut placer un certain dessin et le texte qui l'entourera mais sans se soucier de savoir où se feront les coupures de lignes et quand les lignes auront de nouveau toute la largeur de la gage à leur disposition.

Juxtaposition du code source et du résultat

Les exemples présentant un source et son résultat (soit côte à côte, soit l'un en-dessous de l'autre) ont été faits de façon automatique grâce à un environnement d'un package particulier. Le texte n'a été tapé qu'une seule fois de telle sorte que le résultat montré est forcément celui obtenu par le source en vis-à-vis.

Tableaux

Les tableaux 9.1 et 9.2 page 88 auraient été très pénibles à taper si l'auteur n'avait pas créé quelques macros pour se débarrasser d'une bonne partie du travail répétitif. Cela devrait d'ailleurs devenir un réflexe lorsqu'on travaille sous LaTeX : tout travail répétitif devrait faire l'objet d'une ou de plusieurs macros (ou environnements).

11.2 Graphiques

Initialement, TeX avait été pensé pour produire des ouvrages mathématiques et informatiques. Par conséquent les possibilités graphiques natives sont extrêmement limitées. D'autre part, à l'heure actuelle, les graphiques évoluées font la part belle au format PostScript et, depuis encore moins longtemps au format PDF, formats qui n'existaient pas lorsque Knuth a programmé le logiciel TeX. En fait, cela montre les extraordinaires possibilités de ce logiciel puisqu'il a été possible de construire des extensions permettant d'utiliser ces formats.

[1] C'est à dire dans la version de Jean-Côme Charpentier.

11.2.1 Deux choix possibles

J'ai trouvé la voie !

Il y a deux voies possibles :
- construire une image grâce à un programme externe et l'inclure sous forme d'un fichier PostScript (PDF TEX et PDFLATEX offrent plus de possibilités sur le type des fichiers graphiques pouvant être inclus) ;
- utiliser des extensions permettant de mettre des commandes graphiques dans le source du document.

Ces deux méthodes présentent chacune des avantages et des inconvénients.

Inclusion d'images externes

Inclusion d'images postscripts

Le format le plus " natif " a utiliser sous LATEX est le postscript.

Il faut charger l'extension `\usepackage{graphics}`[2] ou l'extension `\usepackage{graphicx}`[3].

Le fichier image doit être au format " postscript encapsulé " et l'extension du fichier doit être " .eps ".

La commande à utiliser est `\includegraphics[options]{Chemin_acces_du fichier\Nom_Fichier}`.

Précisions sur les chemins d'accès

Le chemin d'accès au fichier peut être absolu ou relatif et la syntaxe dépend du système opératoire utilisé.

Ce qui signifie (en français de France) que je peux écrire :
- `\includegraphics{intro/stallman.ps}`
- ou `\includegraphics{/home/yves/docs/.../cours/latex/Ateliers_urem/cours/intro/stalln`

La première méthode est clairement plus facile. Elle permet aussi de déplacer tout un projet (comme ce cours) sans devoir modifier à la main toutes les commandes d'inclusion de fichier.

Systèmes opératoires et chemins d'accès

Le système opératoire (ou " operating system "), c'est le programme (ou plutôt les programmes) qui fait fonctionner votre ordinateur et qui permet à tous les autres programmes de fonctionner.

Exemples :
- MacOsX,
- Les divers Windows,
- Dos,
- les Unix's et Linux's,
- et plein d'autres.

Sous Dos et Windows, la différence est l'utilisation de la contre oblique comme séparateur de nom de dossier.
- Linux : `\includegraphics{intro/stallman.ps}`
- Dos : `\includegraphics{c:\ex_LaTeX_1\stallman.ps}`

[2] Qui est purement LATEX et postscript.
[3] Qui est plus moderne et fonctionne aussi avec du PDF.

Inclusion d'images PDF, PNG, JPEG

Avec " pdflatex ", d'autres formats d'images peuvent (et doivent) être utilisés. Ceci est cependant générateur d'autres problèmes.

Création d'images externes : les bons programmes

En vrac : xfig, gnuplot, scilab, geogebra ...

Ces programmes sont tous capables de générer directemet du postscript encapsulé utilisable donc directement sous LATEX.

Ils peuvent aussi parfois être appelé par LATEX et les commandes spécifiques sont alors inclues dans le code LATEX moyennant le chargement d'une extension et un environnement spécifique.

xfig

Xfig n'est pas très " joli " mais il permet de créer des images avec du texte et ce texte sera automatiquement dans la police de caractère de votre document.

Création d'images externes : problèmes de cadres

Le postscript encapsulé présente à LATEX des informations sur la taille de l'image. LATEX en général attend cette information puisque l'image étant un " flottant ", LATEX doit savoir comment la placer.

Si l'image n'est pas du postscript encapsulé, LATEX va se plaindre de la " bounding box "[4] manquante.

Une solution est le programme " `ebb` " qui va créer un fichier de " bounding box ". Il faut alors transmettre ce fichier à LATEX.

Des commandes dans le code source

Retour aux sources

Une série d'extensions existent qui permettent d' inclure du code source " spécial " dans le code source LATEX.

PsTricks

Le plus riche mais il pose des problèmes avec la production de PDFs sous " pdfLATEX "

Pgf Tikz

Le préféré de l'UREM ! Voir : `http://sourceforge.net/projects/pgf/`

Des exemples : `http://www.texample.net/tikz/`

Pgf/Tikz peut faire appel à gnuplot comme nous le verrons plus loin et profiter des capacités de calcul de ce dernier.

Pgf Tikz : exemples

[4] C'est à dire du cadre de l'image.

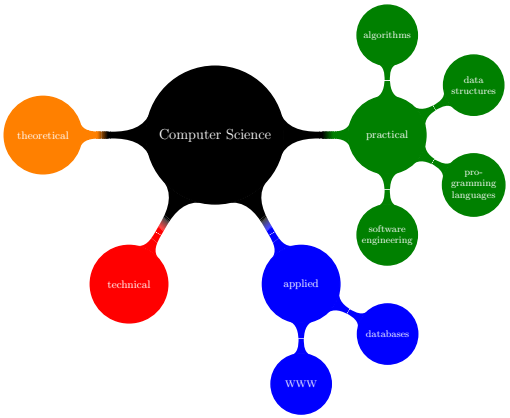


FIG. 11.1 – Carte heuristique avec Pgf/Tikz

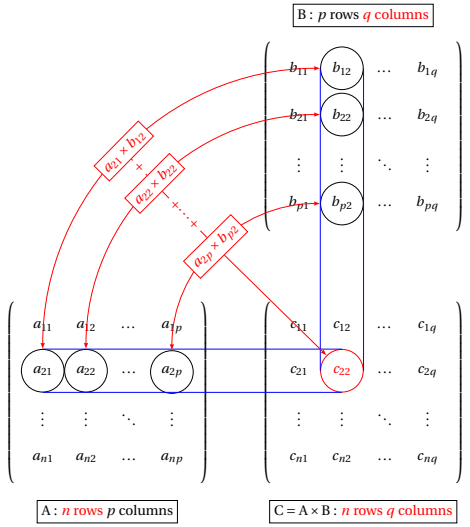


FIG. 11.2 – La multiplication de matrice illustrée avec Pgf/Tikz

Graphismes avec appel à des programmes extérieurs

Une autre voie consiste à faire appel à des programmes annexes qui produisent des fichiers pouvant être inclus dans un source LaTeX, soit sous forme d'images PostScript (ou d'autres formats avec PDFLaTeX), soit directement sous une forme directement compréhensible par LaTeX. Là aussi, il n'est pas possible de réaliser une présentation exhaustive, on pourra citer les programmes utilisés au moins une fois par l'auteur : Gnuplot (tracés de courbes), Scilab (mathématique formelle) , Xfig (dessin vectoriel), METAPOST (comme METAFONT mais produisant des figures PostScript au lieu des fontes), etc.

Il faut, en général passer des arguments supplémentaire à LaTeX comme `-enable-write18` ou `-shell-escape` et compiler deux fois.

gnuplot

Gnuplot a servi pour réaliser le graphique de fonction dans l' exemple d'examen de mathématique de l'atelier II.

Il faut charger l'extension `\usepackage{gnuplottex}` et enfermer les commandes gnuplot dans un environnement.

Voici le code utilisé :

```
\begin{gnuplot}
 f(x)=-1+0.5*2**(x)
 set xrange [-1:3]
 set label "0"  at first  -0.15, first  -0.2
 set xzeroaxis lt 5
 set yzeroaxis lt 5
```

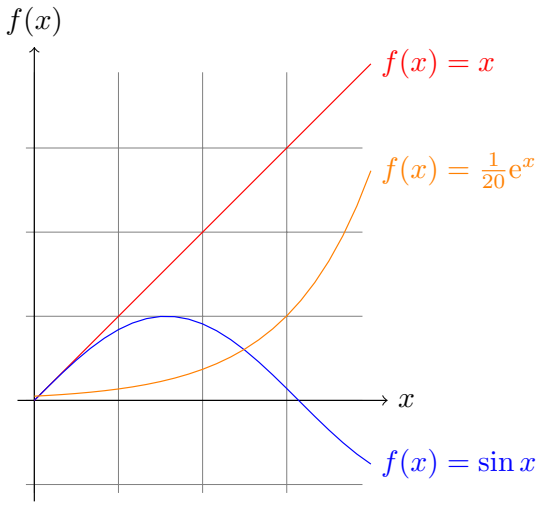
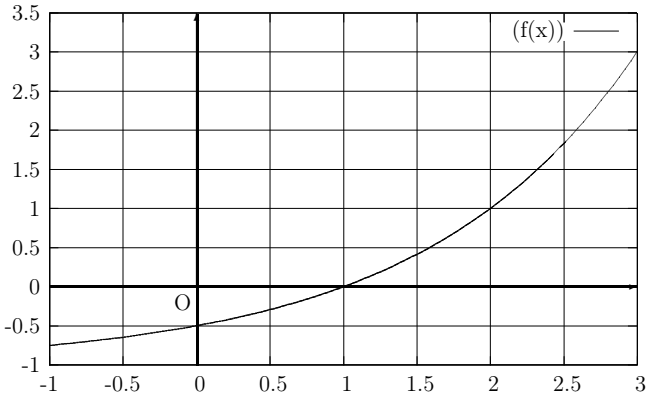


FIG. 11.3 – Graphes de fonctions grâce à Gnuplot et Tikz

```
set arrow from 0,0 to 3,0 ls 4
set arrow from 0,0 to 0,3.5
set grid
plot (f(x))
\end{gnuplot}
```

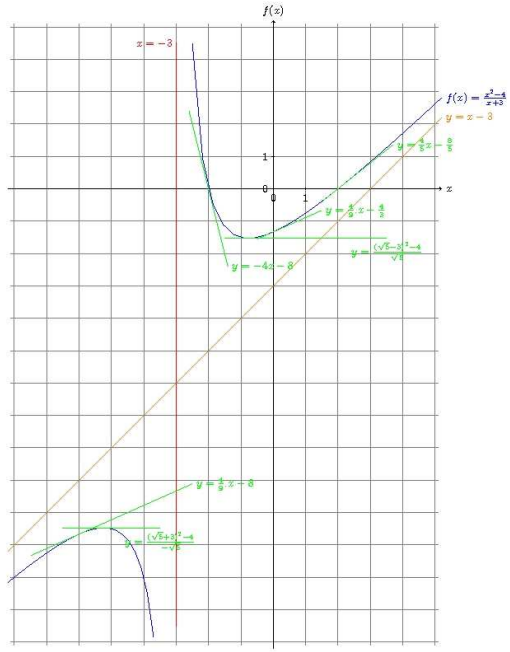


Gnuplot et Pgf/Tikz

Pgf/Tikz peut faire appel à gnuplot pour tracer des courbes de fonctions plus jolies.

Extrait de `http://www.yvesdelhaye.be/?Etude-de-fonction-avec-LaTeX-Giac`

13 Le graphique



11

Metapost

Dérivé de Metafont et adoré par certains.

Voir : `http://fr.wikipedia.org/wiki/MetaPost`

Asymptote

Inspiré de Metapost mais avec un syntaxe à la C++.

Voir : `http://asymptote.sourceforge.net/`

TeXGraph

TEXGraph mérite une mention spéciale. Vous réalisez un dessin et TeXGraph va générer le code correspondant soit en `pstricks` soit en `tikz` ou au choix en :

- LATEX,
- postscript,
- pdf,
- et bien d'autres

Voir :`http://texgraph.tuxfamily.org/`

11.2.2 L'environnement "figure"

Faire bonne "figure"

Il y a un environnement important pour gérer les images dans un texte, c'est l'environnement "figure".

Voici un exemple :

```
\begin{figure}
   \includegraphics{Graphisme/images/computer-science-mindmap.pdf}
 \caption{Carte heuristique avec Pgf/Tikz}
 \label{tikzmindmap}
\end{figure}
```

- Cet environnememnt va donner automatiquement un numéro à la figure.
- La commande `\listoffigures` à la fin du document va créer automatiquement un liste des figures.
- La commande (optionnelle) `\caption{texte sous la figure}` va, elle, ajouter un texte.
- Et il est également possible d'ajouter un "label" pour faire automatiquement référence à la figure et à son numéro de page ailleurs dans le texte.

11.2.3 Dessins scientifiques

Comme LATEX a été très vite employé par l'ensemble des scientifiques, des extensions diverses et variées ont fleuri pour permettre des compositions faciles de schémas dans différentes disciplines.

Il est hors de question de montrer toutes les possibilités. (le lecteur intéressé pourra se reporter à [LGC].

Des extensions permettent ainsi de construire des graphes, des arbres généalogiques ou syntaxiques ou ..., des nœuds, des molécules (avec toute la panoplie de molécules cycliques, de liaisons diverses, ...), des diagrammes de Feynman, des diagrammes de cycles pour les ordinateurs, des schémas optiques, des schémas électroniques, des circuits analogiques ou numériques. Voir [LGC] pour les différentes possibilités.

Chimie

L'extension `ppchtex` permet de dessiner des molécules organiques. Il n'est pas trivial à utiliser.

voir :`http://dev.ulb.ac.be/urem/Dessiner-des-molecules-avec-LaTeX`

11.2.4 Jeux

Les jeux nécessitent des compositions très spécialisées qui doivent suivre des règles strictes pour être facilement lisibles de la part d'un habitué (présentation d'une partie, diagrammes divers, ...). Des extensions permettent de gérer parfaitement les échecs, les échecs chinois, le go, le backgammon, les jeux de cartes (bridge par exemple), les mots croisés ou fléchés. Tous ces exemples ont été pris dans [LGC]

Chapitre 12

Macros et compteurs

12.1 Macros

12.1.1 Principe des macros

Macro = macro-commande

Nous avons étudié un certain nombre de macros jusqu'à maintenant et nous pouvons nous faire une idée générale de ce qu'est une macro : une commande permettant des actions complexes. En réalité, la signification du terme " macro " est un peu plus précise : une macro est un mini-programme construit à partir de commandes directement compréhensibles par le compilateur (en l'occurrence TEX) et/ou d'autres macros définies préalablement. Ainsi, un utilisateur pourra vouloir se construire ses propres macros pour automatiser certaines tâches répétitives.

Un exemple pour comprendre

Reprenons un des exemples précédents où nous avions écrit la formule :

$$\vec{x} \stackrel{\text{déf}}{=} (x_1, x_2, \ldots, x_n)$$

et supposons que le symbole " $\stackrel{\text{déf}}{=}$ " soit souvent utilisé dans le document. Il y aura tout intérêt à définir une macro qui se charge de recopier la définition de ce symbole au lieu de retaper à chaque fois tous les ordres nécessaires à son obtention.

Nommer la macro

La première étape va être de choisir un nom (les `\tata`, `\titi` et autre `\toto` sont à proscrire sans hésitation).

Appelons notre macro `\defeq` pour des raisons de cohérence avec les autres macros de LATEX.

Définir la macro

Ensuite le texte que la macro doit remplacer doit être bien compris et on doit être sûr de son résultat.

Ici, pas de problème, la macro doit remplacer la suite :

```
\stackrel{\mathrm{d\acute ef}}{=}
```

Il ne reste alors plus qu'à appeler la macro permettant de construire des macros, en l'occurrence `\newcommand`, suivie du nom de la macro que l'on veut définir puis d'un groupe donnant son contenu.

Avec notre exemple, cela va donner :

```
\newcommand{\defeq}{\stackrel{\mathrm{d\acute ef}}{=}}
```

Et voilà, c'est tout !

Il ne reste plus qu'à l'utiliser :

$$\vec{x} \stackrel{\text{déf}}{=} (x_1, x_2, \ldots, x_n)$$

```
1  \[\vec x\defeq(x_1,x_2,\ldots,x_n)\]
```

Attention aux noms de macro

Il y a d'autres façons de définir des macros mais celle exposée ci-dessus est fortement conseillée par les défenseurs de LaTeX. Elle offre, entre autres avantages, la sécurité de ne pas redéfinir des macros déjà existantes.

En reprenant l'exemple précédent, on pourrait légitimement se dire que plus un nom de macro est court, plus rapide sera sa frappe et décider d'appeler `\def` notre jolie macro. Si cela avait été possible, il y aurait eu une série de catastrophes et plus rien n'aurait fonctionné correctement car `\def` est une primitive du langage TeX abondamment utilisée (par exemple, la macro `\newcommand` utilise cette primitive) et tout ce qui ferait appel directement ou indirectement à `\def` serait corrompu.

D'autre part, les messages d'erreurs affiché par le terminal auraient été réellement abscons et sans rapport évident avec la redéfinition de la primitive donc une erreur très délicate à comprendre.

12.1.2 Macros à paramètres

Paramètres

Les macros comme celle définie à la section précédente seraient assez utiles mais la puissance de ce concept vient surtout de la possibilité de mettre des paramètres (de 1 à 9).

Illustrons cela par un exemple un peu plus évolué. Supposons qu'à plusieurs endroits d'un ouvrage on veuille écrire des citations avec le texte composé au fer à droite et en italique suivi, à la ligne, du nom de l'auteur composé en petite capitale et enfin du titre de l'ouvrage séparé du nom de l'auteur par une virgule, le tout placé dans un bloc de 6 cm de large.

Exemple de macro à paramètres

Voici un petit exemple pour voir sa réalisation sans définir de nouvelle macro :

```
\begin{flushright}
  \begin{tabular}{@{}p{6cm}@{}}
    {\raggedleft \itshape C'est parce que quelque chose des objets
      ext\'erieurs p\'en\'etre en nous que nous voyons les formes
      et que nous pensons.\par}\\
    {\raggedleft \textsc{\'Epicure}, lettre \'a H\'erodote.\par}
  \end{tabular}
\end{flushright}
```

C'est parce que quelque chose des objets extérieurs pénètre en nous que nous voyons les formes et que nous pensons.

Épicure, lettre à Hérodote.

Petite cerise sur le gâteau : la largeur du texte écrit est exactement de 6 cm car on a pris la peine d'inhiber les espaces intercolonnes en mettant les deux `@{}` au début et à la fin du motif du tableau.

Le tout est un peu embêtant à taper et nous voudrions avoir une macro qui se charge du travail, étant entendu que le texte de la citation, le nom de l'auteur et le titre de l'ouvrage sont variables. Pour cela, il va falloir définir une macro à trois paramètres.

Déclaration de la macro à paramètres

Appelons-la `\epigraphe`, le début de sa déclaration se fait sous la forme :

```
\newcommand{\epigraphe}[3]
```

c'est-à-dire comme précédemment mais en rajoutant le nombre de paramètres entre crochets.

Paramètre = dièse

Ensuite, on tape le texte qui remplacera la macro en indiquant la présence du premier paramètre par la séquence `#1`, celle du deuxième paramètre par `#2` et celle du troisième paramètre par `#3`.

Comme nous avons déjà le modèle, cela ne va pas poser de problème insurmontable, il suffit de remplacer les passages correspondants par les séquences `#`n adéquates :

```
\newcommand{\epigraphe}[3]{
  \begin{flushright}
    \begin{tabular}{@{}p{6cm}@{}}
      {\raggedleft \itshape #1\par}\\
      {\raggedleft \textsc{#2}, #3\par}
    \end{tabular}
  \end{flushright}}
```

L'emploi est on ne peut plus simple, il suffit d'écrire la macro puis les trois éléments (dans l'ordre) et le tour est joué. Par exemple :

```
\epigraphe{Voici une macro qui parle d'elle-m\^eme
 (en logique, on appelle cela un quine).
Une macro compliqu\'ee qui fonctionne
du premier coup est un bonheur rare.}
{Charpentier}
{Stage \LaTeX}
```

Voici une macro qui parle d'elle-même (en logique, on appelle cela un quine). Une macro compliquée qui fonctionne du premier coup est un bonheur rare.

Charpentier, Stage LaTeX

Cuisine interne

L'emploi de macros personnelles ne sert pas qu'à remplacer des séquences longues et répétitives de commandes, il permet également de réaliser des documents faciles à maintenir car mieux structurés. Supposons qu'on réalise un document produisant des recettes de cuisines. Chaque recette va commencer avec le nom du plat qu'on écrira dans un corps plus grand que le reste de la fiche.

Il n'y a pas de problème particulier, un source tel que :

```
{\Large Le canard laqu\'e}
```

conviendra parfaitement.

Après des semaines de travail, l'ouvrage comporte maintenant plus de 400 fiches lorsque, tout compte fait, les noms des recettes seraient plus jolis s'ils étaient aussi en gras et centrés en haut de la fiche, c'est-à-dire que chaque fiche devrait commencer sur une nouvelle page. Bien évidemment, les éditeurs de texte sont capables de faire des remplacements automatiques mais il s'agit presque toujours d'opérations dangereuses et qui doivent être effectuée sous contrôle humain ; après tout, d'autres textes ont pu être composés dans ce même corps sans être le nom de la recette.

Une solution beaucoup plus propre et compréhensible serait de définir dès le début du travail une macro `\titrerecette` dont la déclaration serait :

```
\newcommand{\titrerecette}[1]{{\Large #1}}
```

La présence des doubles accolades est nécessaire car sinon, l'appel `\titrerecette{Le canard laqu\'e}` est remplacé par le texte :

```
\Large Le canard laqu\'e
```

et l'action de la macro `\Large` se poursuivra au-delà de la zone souhaitée alors qu'avec les doubles accolades, la macro est remplacée par le texte :

```
{\Large Le canard laqu\'e}
```

et tout va bien.

Au moment où on veut modifier la présentation des titres des recettes, il suffira de modifier la définition de la macro au niveau du préambule et tout sera recomposé selon les nouvelles exigences. Ici, la macro deviendrait :

```
\newcommand{\titrerecette}[1]{%
  \newpage
  \begin{center}
    \Large #1
  \end{center}
  }
```

Deux petits mots sur cette macro avant de clore le sujet. Nous avons vu précédemment qu'il fallait mettre la macro `\Large` entre accolades pour que son action reste confinée au paramètre de la macro. Ici, ce n'est pas nécessaire car tout environnement se comporte comme des accolades sur ce point. En l'occurrence, l'environnement `center` va limiter la portée de la macro `\Large`. D'autre part, il y a un caractère `%` à la fin de la première ligne. Ce commentaire (vide) sert à ce que TEX ne voit pas la fin de la ligne et ne produisent donc pas d'espace parasite. En fait, cette précaution est ici superflue car une espace précédant un saut de page ne fait de mal à personne mais il existe beaucoup de situations où il faut penser à ce " truc ".

12.1.3 Compteurs et dimensions

Registres

Dans ce manuel, nous avons déjà rencontré les deux dimensions `\parindent` et `\parskip` qui permettaient respectivement d'indiquer l'importance de l'indentation de première ligne et la distance entre deux paragraphes. Nous avons également utilisé, sans le nommer, le registre qui garde le numéro de page en cours. En fait, LATEX gère une multitude de choses en se servant de registres de dimension et de compteurs ; de plus, l'utilisateur peut se créer ses propres registres pour des besoins particuliers.

Compteurs

Étudions d'abord les compteurs, la syntaxe pour les dimensions ne sera pas trop différente. Si on désire utiliser un compteur qui n'est pas défini par LATEX, il va falloir le déclarer. Pour cela, on dispose de la macro `\newcounter` qui sera suivie du nom du compteur (compteur qui, sous LATEX, ne s'écrit pas avec la contre-oblique). Nous allons déclarer un compteur qui servira à une macro destinée à numéroter automatiquement les exercices d'un devoir ; appelons-le `numexo`. Sa déclaration sera donc :

```
\newcounter{numexo}
```

Cette déclaration faite, le compteur `numexo` existe et est initialisé à zéro. Il reste à pouvoir modifier sa valeur, à le transmettre à des macros et à placer sa valeur dans un texte. Voyons ces différents points un à un. Avec LATEX, il existe deux façons de changer le contenu d'un compteur. On peut placer une valeur particulière grâce à la syntaxe :

`\setcounter{`*cpt*`}{`*val*`}`

qui permet de mettre la valeur *val* dans le compteur *cpt*.

On peut aussi ajouter une valeur en utilisant la construction :

`\addtocounter{`*cpt*`}{`*val*`}`

qui ajoutera la valeur *val* au contenu actuel du compteur *cpt*. Pour une soustraction, il suffit d'ajouter une valeur négative. Pour les multiplications et divisions, vous êtes cordialement invité au stage de perfectionnement ou à lire des documents de référence !

Si le contenu d'un compteur doit être transmis à une macro en tant que paramètre, on pourra utiliser la construction :

`\value{`*cpt*`}`

Un point plus utile que le précédent consiste à afficher le contenu d'un compteur au niveau du document produit. C'est ce que nous voulons avec le compteur d'exercice. Pour cela, LATEX n'offre pas moins de 7 macros.

Nous ne verrons ici que les cinq les plus fréquemment employées :

Commande	Type	Exemple
`\arabic{`*cpt*`}`	nombre arabe	1, 2, 3, ...
`\roman{`*cpt*`}`	nombre romain minuscule	i, ii, iii, ...
`\Roman{`*cpt*`}`	nombre romain majuscule	I, II, III, ...
`\alph{`*cpt*`}`	lettre minuscule	a, b, c, ...
`\Alph{`*cpt*`}`	lettre majuscule	A, B, C, ...

Nous sommes maintenant en possession de tout ce qu'il faut pour construire notre macro gérant automatiquement les numéros d'exercices. Cette macro devra définir un nouveau paragraphe, inhiber l'indentation de première ligne et afficher le texte " **Exercice** n. " où n sera le numéro de l'exercice en cours suivie d'une espace horizontale assez généreuse.

Le code réalisant toutes ces actions peut être défini comme suit :

```
\newcommand{\exo}{\par
  \addtocounter{numexo}{1}%
  \noindent
  \textbf{Exercice \arabic{numexo}}\quad}
```

À la suite de quoi on pourra obtenir quelque chose comme :

```
\exo Rep\'erer toutes les fautes de ce manuel
(exercice difficile et long).
\exo Assez t\^ot dans ce manuel, il avait \'et\'e dit que
le nombre de macros utilis\'ees jusqu'\`a ce point \'etait de~78.

Maintenant que ce manuel arrive plut\^ot vers sa fin, quel
est le nombre de macros qui ont \'et\'e \'etudi\'ees ?
\exo Tr\`es souvent, l'augmentation d'un compteur avec
la macro \(\mathtt{\backslash}\)\texttt{addtocounter} est
 de~1 (comme pour l'exemple en cours des exercices).

Pensez-vous vraiment que \LaTeX{} soit si mal fait qu'il
n'ait pas pens\'e \`a d\'efinir un raccourci pour une incr\'ementation d'une unit\'e ?
```

Exercice 1 Repérer toutes les fautes de ce manuel (exercice difficile et long).

Exercice 2 Assez tôt dans ce manuel, il avait été dit que le nombre de macros utilisées jusqu'à ce point était de 78.

Maintenant que ce manuel arrive plutôt vers sa fin, quel est le nombre de macros qui ont été étudiées ?

Exercice 3 Très souvent, l'augmentation d'un compteur avec la macro `\addtocounter` est de 1 (comme pour l'exemple en cours des exercices).

Pensez-vous vraiment que LaTeX soit si mal fait qu'il n'ait pas pensé à définir un raccourci pour une incrémentation d'une unité ?

Facile n'est-ce pas ? Et surtout, cela évite les erreurs de numérotations qui arrivent fréquemment surtout si on remanie l'ordre et le nombre d'exercices en étant un peu pressé (expérience personnelle).

À propos de l'exercice 3 : la réponse est évidemment " non " ! Il arrive effectivement très fréquemment que l'on veuille ajouter 1 à un compteur et la séquence :

`\addtocounter{`*cpt*`}{1}`

peut être raccourcie en :

`\stepcounter{`*cpt*`}`

Compliquons un peu les choses en exigeant une macro qui numérote automatiquement les questions à l'intérieur d'un exercice mais qui ait le bon goût de redémarrer à 1 à chaque nouvel exercice.

Pour que cela soit joli, le numéro de la question sera écrit en gras et suivi d'un triangle, d'un point et d'une espace normale. Nous appellerons `\question` cette macro et `numq` le compteur associé au numéro de question.

La définition de la macro `\question` ne pose pas de problème : elle ressemble beaucoup à celle de la macro `\exo` :

```
\newcommand{\question}{\par
  \stepcounter{numq}%
  \noindent
  \textbf{\arabic{numq}} \(\triangleright\).\ }
```

Avec les deux macros `\exo` et `\question` telles qu'elles ont été définies, les choses seront correctes pour le premier exercice mais deviendront incorrectes dès le deuxième exercice car le compteur `numq` continuera sur sa lancée.

Il existe deux solutions à ce problème : soit on modifie la définition de la macro `\exo` en ajoutant la commande :

`\setcounter{numq}{0}%`

au début de la macro, soit on utilise une autre façon de définir le compteur `numq` :

`\newcounter{numq}[numexo]`

Cette syntaxe indique que `numq` est un compteur qui sera automatiquement remis à zéro à chaque appel de `\stepcounter{numexo}` (il faudra donc obligatoirement modifier la macro `\exo` pour y remplacer la commande `\addtocounter{numexo}{1}` par la commande `\stepcounter{numexo}`).

Certaines macros gérant les dimensions ressemblent à celles dédiées aux compteurs. Ainsi, on trouve :

`\newlength{\`*long*`}`

qui permet de définir la dimension *long* (on notera la présence de la contre-oblique pour les dimensions alors qu'il n'y en avait pas pour les compteurs). Lorsqu'une nouvelle dimension est créée, elle est initialisée à `1in` (1 pouce).

On a également :

`\setlength{\`*long*`}{`*val*`}`

qui permet de spécifier la valeur *val* à placer dans le registre de dimension *long*. Une longueur doit obligatoirement comporter une des unités listées au tableau 7.3 page 71 et peut comporter une composante `plus` et/ou une composante `minus` permettant de donner respectivement un étirement et une compression à cette longueur (TeX parle plutôt de ressort), Cf. section 7.0.6 page 72.

Enfin, la dernière macro ressemblant à celles gérant les compteurs est :

`\addtolength{\`*long*`}{`*val*`}`

qui ajoute la valeur *val* au registre *long*. Les trois composantes (longueur, extension et compression) sont ajoutées séparément.

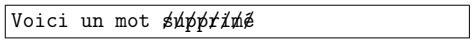
Trois autres macros permettent de spécifier une longueur calculée à partir d'un matériel donné :

`\settowidth{\`*long*`}{`*objet*`}`
`\settoheight{\`*long*`}{`*objet*`}`
`\settodepth{\`*long*`}{`*objet*`}`

initialisent le registre *long* respectivement avec la largeur, la hauteur et la profondeur de *objet*.

Exemple

Un petit exemple pour bien comprendre qui utilise le fait que la fonte `\texttt{}` a tous ses caractères de la même largeur :




```
\newlength{\retour}
\settowidth{\retour}{\texttt{supprim\'e}}
\texttt{Voici un mot
  supprim\'e\hspace{-\retour}////////}
```