LaTeX, un peu, beaucoup

Pascal Dupont

Mots clés : LaTeX, programmation, traitement de données

11. Multitraitement

À vrai dire, je ne sais pas trop si le titre choisi pour cette rubrique est bien adéquat — déjà, c'est un néologisme. Évoque-t-il seulement quelque chose pour le lecteur ? Expliquons plutôt quelle est l'idée qui va être développée.

Je dispose d'un texte ou d'un ensemble de données, qui doit être mis en forme pour constituer un document bien présenté. Ou plutôt, je voudrais pouvoir élaborer différents documents à partir du même texte ou du même ensemble de données, en variant leur présentation. C'est ici qu'il est bon de se souvenir que TeX est, avant tout, un langage de programmation.

Voici trois exemples qui, je l'espère, clarifieront mon propos.

1. Je prépare, pour mes étudiants, des listes d'exercices. Eux recevront la liste des énoncés, mais je souhaiterais, pour mon usage personnel, disposer d'une version où chaque problème serait accompagné de sa réponse, pour que je puisse procéder facilement à la vérification du travail de mes étudiants.

2. J'ai reçu, de l'administration de mon établissement, la liste de mes étudiants ; cette liste est fournie dans un « classeur » lisible par un tableur et comprend un certain nombre d'informations sur chaque étudiant ; elle est accompagnée d'un dossier de photos. Je voudrais imprimer l'information sous la forme d'une simple liste alphabétique des noms et prénoms, mais aussi disposer d'une liste illustrée des photos et indiquant pour chacun, outre le nom et le prénom, le numéro matricule et le code de la filière d'études.

3. J'utilise, pour mon cours, des « dias » informatiques. Outre la version que je projette, je fournis à mes étudiants une version imprimable reprenant quatre dias par page A4 ; en outre, je voudrais disposer, pour aller donner cours, d'une version imprimée, avec deux dias par page et, en regard de chacune, des indications sur les commentaires que je ne dois pas oublier de faire, les exemples que je vais montrer au tableau, &c.

On le voit, il s'agit presque d'utiliser LaTeX comme un gestionnaire de bases de données ; « presque », parce qu'il ne dispose pas, notamment, de fonctions de tri. Prévoir des critères de sélection, cependant, est possible. Le lecteur pourra adapter les idées présentées ici à d'autres situations, en fonction de ses besoins.

Listes d'exercices

Le problème des listes d'exercices, bien sûr, admet une solution élémentaire : je tape ma liste en faisant suivre chaque exercice de sa réponse, dans un document que j'appelle, p. ex., *Exercices_E+S.tex*. Ensuite, je crée une copie de ce document sous le nom de *Exercices_E.tex* et, dans ce nouveau document, je supprime les réponses ou, mieux, je les mets en commentaire en les faisant précéder du caractère « % ».

Cette « solution » est cependant fortement boiteuse, parce qu'un texte de ce type n'est jamais gravé dans le marbre pour l'éternité : très rapidement, j'aurai à y faire des adaptations, des corrections, des additions, des modifications de l'ordre, et que sais-je encore. Or, ces mises à jour sont, chaque fois, à effectuer dans les deux documents, de manière parfaitement coordonnée ! Chacun se rendra rapidement compte que ceci n'est pas tenable.

Voici donc l'ébauche d'une meilleure solution. Dans un document nommé *Questions.tex*, je range mes exercices sous la forme de macros : `\exer{`*`Enoncé`*`}{`*`Réponse`*`}`. Le document destiné aux élèves, *Exercices_E.tex*, contiendra alors, en substance, ceci :

```
\documentclass{article}
\newcommand{\exer}[2]{#1}
\begin{document}
\input{Questions.tex}
\end{document}.
```

De son côté, le document source pour *ma* version de la liste sera :

```
\documentclass{article}
\usepackage{xcolor}
\newcommand{\exer}[2]{#1\\
\textcolor{blue}{\textsc{Rép. :} #2}}
\begin{document}
\input{Questions.tex}
\end{document}.
```

Ainsi, pour chaque exercice de la liste, l'énoncé sera imprimé ; puis, à la ligne, la réponse sera donnée, précédée de « Rép. : », le tout écrit en bleu pour bien se distinguer de l'énoncé. Au contraire, dans le document destiné aux élèves, le second argument de chaque macro, à savoir la réponse, était tout bonnement ignoré.

Il se pourrait aussi que je me ravise et que je décide de donner aux élèves un document comprenant les énoncés et les réponses, mais dans deux parties séparées, pour qu'ils n'aient pas la réponse sous les yeux en lisant l'énoncé d'un problème. Bonne nouvelle : il n'est nullement besoin de retravailler le fichier *Questions.tex* ; il suffira simplement que le document principal qui l'appelle soit rédigé comme suit :

```
\documentclass{article}
\newcommand{\exer}[2]{#1}
\begin{document}
\begin{center}
\Large\textbf{Énoncés}
\end{center}
\input{Questions.tex}
\pagebreak
\renewcommand{\exer}[2]{#2}
\begin{center}
\Large\textbf{Réponses}
\end{center}
\input{Questions.tex}
\end{document}.
```

En deux mots : le document principal lit deux fois la liste des questions, en modifiant entre les deux la définition de la macro `\exer` : au premier passage, seul le premier argument (l'énoncé) est pris en compte, tandis qu'à la seconde lecture, la macro ignore son premier argument pour n'imprimer que le second (la réponse). Pour faire bonne mesure, j'ai ajouté à cela l'impression des titres « Énoncés » et « Réponses », et un saut de page entre les deux parties.

Ainsi, même si énoncés et réponses sont séparés dans le document compilé, ils sont groupés dans le document source, de sorte que, lorsque je souhaite modifier l'ordre des exercices ou ajouter une variante de l'un d'entre eux, les manœuvres de copier-coller à effectuer sont beaucoup moins risquées que s'il fallait les pratiquer deux fois, dans les énoncés d'abord et dans les réponses ensuite.

Ce qui précède est évidemment schématique, simplifié pour mettre en évidence le principe. Les commandes telles que je les utilise en réalité sont plus sophistiquées que celles que j'ai décrites ci-dessus, afin notamment de faire numéroter automatiquement les exercices de la liste — et leurs réponses, ce qui est indispensable pour pouvoir rattacher les réponses aux exercices lorsque les deux sont séparés.

En fait, c'est une commande à *trois* arguments que j'utilise pour mes exercices : énoncé, réponse et solution développée. Je peux alors compiler la liste d'exercices de manière à avoir un exercice par feuille, chacun accompagné de sa résolution détaillée. Un test vérifie alors si le troisième argument est bien présent, et remplace la solution développée par la réponse finale dans les cas où je n'ai pas (encore) eu le courage de taper la résolution.

Cette structure ternaire, imaginée il y a plus de vingt ans maintenant, a largement fait la preuve de sa souplesse : je l'utilise dans des contextes très variés depuis tout ce temps, sans avoir jamais dû la remettre en cause. Elle me permet aussi, par exemple, de produire mes questionnaires d'examen, puis d'en publier les corrigés.

J'utilise aussi une macro pour regrouper des exercices (typiquement, du drill) sous un chapeau commun, comme les exercices 128–131 dans ce petit extrait :

Exemple

127. Soit $f : x \mapsto x^2 + 4x - 3$; calculer $f'(-1)$ en utilisant la définition.

▷ Dériver les fonctions $f : x \mapsto$
128. $\ln(2x + 3)$;
129. $(x^2 + x + 3)^4(x^2 - 4x + 5)^3$;
130. $x^2 e^{2x}$;
131. x^{x^x}

(présenter la dérivée sous forme factorisée).

132. Calculer la dérivée seconde de $x \mapsto e^{-x^2}$.

Exemple

Le nom que j'ai choisi pour la macro dédiée à ce type de sous-liste est `\multi`, avec trois arguments : le « chapeau », la liste d'exercices (en principe, rien que des `\exer{...}{...}{...}`) et le « talon » (vide, le plus souvent).

Listes d'étudiants

L'informatique administrative de mon université me permet d'obtenir, dans mon navigateur, la liste des étudiants inscrits à chacun de mes cours, et ensuite de télécharger cette liste dans un classeur de format *.xls*, avec pour chacun d'entre eux un certain nombre de données : les nom et prénom(s), évidemment, mais aussi leur numéro matricule, leur filière d'études et son code, leur sexe (qui n'est pas nécessairement évident à déterminer sur base du prénom !), &c ([1]).

Je peux également télécharger un dossier contenant leurs photos. Le point intéressant dans l'histoire est que, dans ce dossier, le nom de l'image est *nnnnnnnn.jpg*, où nnnnnnnn est le numéro matricule de l'étudiant ; nous verrons dans un instant comment l'exploiter.

Revenons au classeur ; nombre de ses colonnes ne m'intéressent pas ; je crée donc une nouvelle feuille où je recopie les numéros matricules, noms, prénoms, sexe et code de la filière. Je copie les cellules contenant de l'information et je colle leur contenu dans un document LaTeX. Chaque étudiant occupe une ligne de ce document, et les différents champs, dans chaque ligne, sont séparés par des tabulations. Je demande alors à l'éditeur de texte de remplacer tous les « *tab* » par « `}{` » et tous les « *ret* » (fin de ligne) par « `}`*ret*`\fiche{` ». Ceci ne demande donc que deux opérations manuelles, quel que soit le nombre d'étudiants ; il reste toutefois de petites corrections à apporter à la première et à la dernière ligne du fichier.

Chaque ligne de mon document contient maintenant un texte du type

```
\fiche{20151234}{Honyme}{Anne}{F}
  {GBIGES009901}.
```

J'enregistre ce document comme *CodeCours.tex* (où *CodeCours* ressemble à *MATHnnnn*, mais peu importe).

Il reste à créer le document principal, qui définira la macro `\fiche` puis appellera *CodeCours.tex*. Par exemple, pour une simple liste des noms et prénoms, je créerai le document source *CodeCours_L.tex* avec ce contenu :

```
\documentclass{article}
\newcommand{\fiche}[5]
  {\item\textsc{#2} #3}
\begin{document}
\begin{center}
\Large\textbf{CodeCours --- 2015--2016}
\end{center}
\begin{enumerate}
\input{CodeCours.tex}
\end{enumerate}
\end{document}.
```

Dans un autre document, *CodeCours_P.tex*, la définition de la macro `\fiche` sera plutôt quelque chose du genre

```
\newcommand{\fiche}[5]
\fbox{\begin{minipage}[b]{40mm}
\begin{center}
\includegraphics[width=30mm]
  {\dossier#1.jpg}\\
\textsc{#2}\\
#3 (#4)\\
{\footnotesize#1}\\
{\footnotesize(#5)}\\
\end{center}
\end{minipage}}}.
```

Elle produira donc chaque fois une « boite » encadrée contenant, de haut en bas, la photo de l'étudiant, son nom en petites capitales, son prénom suivi entre parenthèses de « F » ou de « M » selon le sexe, son numéro matricule (en petits caractères) et enfin le code de la filière (en petit, entre parenthèses).

Le lecteur observateur aura noté que l'image incorporée est appelée `\dossier#1.jpg` ; « `#1` » est le premier argument de la macro, soit le numéro matricule de l'étudiant ; et nous avons dit plus haut que la photo, au format JPEG, se trouve dans un document portant pour nom le numéro matricule ; mais que vient faire ici la macro `\dossier` ? Tout simplement, il n'est sans doute pas raisonnable, dans le dossier qui contient déjà nos deux documents *CodeCours.tex* et *CodeCours_P.tex* ainsi

([1]) Il y a aussi une colonne intitulée « Nom de recherche », qui contient le nom débarrassé de ses blancs, traits d'union, apostrophes, &c., ce qui permet de faire un tri alphabétique correct : les tableurs ont en effet encore beaucoup à apprendre sur ce point.

que les *.log* et *.pdf* correspondants, et peut-être encore quelques autres, de laisser en vrac quelques dizaines ou quelques centaines de photos ; il est plus propre, n'est-ce pas, de ranger celles-ci dans un sous-dossier qui sera baptisé, pourquoi pas, *Photos*. Si c'est le cas, je placerai dans le préambule la définition
`\newcommand{\dossier}{Photos/}` ;
ainsi, lorsque la macro `\fiche` convoque la photo de l'étudiant, c'est le document *Photos/nnnnnnnn.jpg* qu'il appelle, c'est-à-dire le document *nnnnnnnn.jpg* du dossier *Photos*. En définissant la macro `\dossier` de la manière adéquate (la syntaxe exacte dépendra du système d'exploitation), il est possible de ranger les photos des étudiants à n'importe quel endroit.

Dans un troisième document, *CodeCours_F.tex*, j'utilise une définition encore différente de la macro `\fiche`, de manière à obtenir des fiches au format A5 (deux par feuille A4, donc), que j'utilise pour prendre des notes durant mes examens oraux.

Dias projetables, dias imprimables

Dans le troisième exemple que voici, le lecteur qui n'a jamais entendu parler de *beamer* risque d'être un peu perdu. Je le prie de m'en excuser ([2]). Je me permets cependant, sans entrer dans les détails, d'en dire quelques mots, car ce troisième exemple, tout en creusant les idées précédentes, en introduit quelques nouvelles.

Pour les dias que j'utilise au cours, ainsi que pour différents exposés, je fais appel à la « classe » *beamer*. (La « classe » d'un document LaTeX, c'est en gros son type : article, livre, &c. ; elle est sélectionnée par la toute première instruction du préambule : `\documentclass{...}`.)

Sans entrer dans tous les détails, pour obtenir la version « à projeter » des dias, le préambule doit, par exemple, commencer par :

```
\documentclass{beamer}
\mode<presentation>{\usetheme{default}}
```

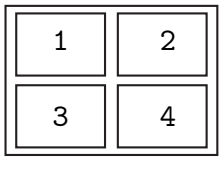
tandis que pour obtenir la version imprimable (4 dias par page, en format « paysage »), c'est :

```
\documentclass[handout]{beamer}
\usepackage{pgfpages}
\pgfpagesuselayout{4 on 1}
  [a4paper,border shrink=2mm]
```

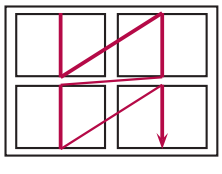
Pour passer d'une version à l'autre, une méthode est d'avoir dans le document, disons *Dias.tex*, les deux jeux d'instructions, et de masquer l'un ou l'autre à l'aide de « `%` » en fonction du résultat souhaité ; mais bien sûr, cela oblige à renommer le document compilé, en *DiasPj.pdf* ou en *DiasIm.pdf* (avec les suffixes *Pj* pour « projection » et *Im* pour « impression »), avant de recompiler pour obtenir la seconde version, sous peine de voir « écrasée » la première. C'est donc une solution assez douteuse...

Il est de loin préférable d'avoir deux documents source différents, *DiasPj.tex* et *DiasIm.tex*. Mais évidemment, nous y revoilà, cela oblige à modifier les deux documents de manière rigoureusement parallèle à chaque fois que nous avons à opérer le moindre changement... attention, danger ! Donc, et ceci rejoint ce qui a été dit plus haut, au sujet des listes d'exercices, la bonne solution est d'avoir des documents *DiasPj.tex* et *DiasIm.tex* qui ne sont à peu près que des coquilles vides : ils se résument, pour ainsi dire, au préambule, puis appellent, l'un et l'autre, le document commun *Dias.tex*, qui contient toute la substance.

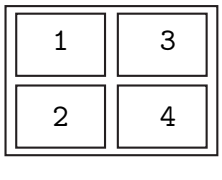
J'ouvre une petite parenthèse. Il n'est pas tout à fait vrai que c'est ainsi que je procède pour obtenir la version imprimable. En effet, lorsqu'on lui demande 4 dias par page au moyen des instructions indiquées ci-dessus, *beamer* les dispose comme suit :



ce qui correspond au sens de lecture

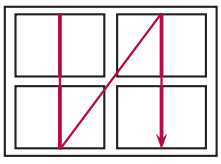


Je préfère, pour ma part, la disposition



([2]) S'il y a une demande pour que cette rubrique présente *beamer*, je suis disposé à le faire : cela occupera bien deux ou trois numéros.

qui donne, plus logiquement, une lecture colonne après colonne :

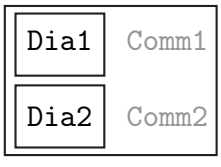


Avec *beamer*, il est impossible de choisir ; il y a heureusement un moyen pour s'en sortir : demander *2* dias par page (il s'agit alors de pages au format « portrait »), avec les instructions

```
\documentclass[handout]{beamer}
\usepackage{pgfpages}
\pgfpagesuselayout{2 on 1}
   [a4paper,border shrink=2mm],
```

puis procéder à une « impression virtuelle » de 2 pages sur 1 ; il s'agit en quelque sorte d'un post-traitement du PDF. Fin de la parenthèse.

Il me reste à expliquer comment je produis la version imprimable avec commentaires à droite de chaque dia :



Il faut savoir que le principe, dans *beamer*, est que chaque dia correspond à un environnement *frame*, balisé donc par les instructions `\begin{frame}` et `\end{frame}`. Sans entrer dans tous les détails, j'ai défini, dans le préambule de mon document *DiasNt.tex* (avec le suffixe *Nt* pour « notes »), un environnement *FRAME* que j'utilise à la place de l'environnement *frame*. Mon environnement *FRAME* produit en fait *deux* dias, la première avec le contenu normal, celui qui doit être projeté, et la seconde avec le contenu d'une macro `\NOTES{...}` ; ainsi,

```
\begin{FRAME}
Dia
\NOTES{Commentaires}
\end{FRAME}
```

sera-t-il interprété à la compilation comme

```
\begin{frame}
Dia
\end{frame}
```

```
\begin{frame}
Commentaires
\end{frame}.
```

Ceci me donne le résultat souhaité lorsque j'imprime mes dias à raison de 4 par page (dans l'ordre standard de *beamer* !).

Pour faire bonne mesure, j'ajoute que j'ai aussi défini deux macros `\NOTESip` et `\NOTESpi`, dont les contenus ne sont imprimés, respectivement, que lors des années académiques « impaire-paire » (comme 2015–2016) ou « paire-impaire » (comme 2014–2015) ; ceci me permet d'avoir deux jeux d'exemples en alternance, de manière que les étudiants qui « approfondissent » n'entendent pas deux fois la même chose.

J'aurais pu, me direz-vous, toujours travailler avec deux dias standard, sans me donner la peine d'introduire ce nouvel environnement *FRAME*. Il n'en est rien. D'abord, il y a un risque supplémentaire de perdre en cours de route une des deux dias « sœurs » lorsqu'on effectue un copier-coller (ci-dessus, dans la vision schématique que j'en donne, tout tient en quatre lignes ; mais une vraie dia de cours peut très bien comprendre plusieurs dizaines de lignes de code, voire plusieurs centaines, de sorte qu'en repérer le début et la fin n'est pas toujours si facile, même avec de bonnes habitudes dans la présentation du document source) ; il y a aussi un petit danger, lorsqu'une dia n'appelle pas de commentaire, d'oublier de lui adjoindre la « petite sœur » vide, ce qui décalerait toute la présentation. Mais il y a une raison beaucoup plus fondamentale, que voici.

Dans le préambule des deux autres documents *DiasPj.tex* et *DiasIm.tex*, l'environnement *FRAME* est tout simplement déclaré synonyme de *frame*, et la macro `\NOTES` ainsi que ses deux variantes sont déclarées inopérantes, de manière que leur contenu soit ignoré. De cette manière, tout se passe de manière transparente lors de la compilation des dias à projeter, ainsi que lors de la production de la version à imprimer par les étudiants.

Dans ce texte, je me suis contenté d'exposer les idées générales. Imprimer de long en large des lignes de code effectivement fonctionnel n'aurait guère de sens ici, mais le lecteur intéressé pourra trouver des exemples tout prêts sur le site de la SBPMef.

Pascal Dupont est chargé de cours à HEC•ULg. ✉ pascal.dupont@ulg.ac.be.