LaTeX, un peu, beaucoup

Pascal Dupont

Mots clés : LaTeX, macros, TeX

15. Les macros « à la TeX »

Une macro, en LaTeX, se définit au moyen de la commande `\newcommand` (et se redéfinit au moyen de `\renewcommand`). Ceci a été expliqué dans le n° 5 de cette rubrique (*Losanges* **26**, 54–57) ; dans le même article, j'indiquais déjà que le TeX « de base » utilisait une syntaxe différente : là où la définition de la macro produisant les intervalles ouverts pourrait être

```
\newcommand{\oo}[2]{\left]#1 ;#2\right[}
```

en LaTeX, en utilisant la syntaxe TeX, ce qui est possible à l'intérieur d'un document LaTeX (puisque LaTeX n'est qu'un ensemble de commandes *ajoutées* au TeX de base, pour en rendre l'usage plus aisé), nous coderions plutôt

```
\def\oo#1#2{\left]#1 ;#2\right[}
```

Outre que le nom de la macro ainsi définie ne doit pas, ici, être entourée d'accolades, le *nombre* d'arguments (qui est l'argument optionnel de `\newcommand` et figure donc entre crochets), ici `2`, est remplacé par leur *liste* `#1#2`. Rappelons que le nombre maximal d'arguments est neuf, que ce soit en LaTeX ou en TeX. Par ailleurs, TeX ne fait pas de distinction entre définir et redéfinir une commande : la commande `\redef` n'existe pas ; ou, pour le dire autrement : TeX ne vérifie pas si vous « écrasez » une commande existante.

L'utilisation de la macro, elle, est exactement la même, quelle que soit la manière dont elle a été définie. Chacun de ses arguments est soit un caractère unique (`a`, `5`, `;`, …), soit un nom de commande sans argument (`\&`, `\infty`, …), soit un groupe délimité par des accolades (`{2017}`, `{\pi^2}`, …). Par exemple, si nous codons (en mode mathématique !) `\oo020`, les deux arguments de la commande « intervalle ouvert » définie ci-dessus sont « 0 » et « 2 », de sorte que nous obtenons $]0;2[$; si notre objectif était d'obtenir $]0;20[$, c'est `\oo0{20}` qu'il convenait de placer dans le document source. Par contre, l'intervalle $]\pi;\pi^2[$ peut se coder `\oo\pi{\pi^2}`, sans accolades autour du premier argument, puisque celui-ci est la commande `\pi`, qui est interprétée par le compilateur comme un objet monolithique ; cependant, placer des paires d'accolades supplémentaires n'est pas interdit — et est parfois recommandable pour augmenter la lisibilité.

Arguments délimités

Ce qui est particulier à la syntaxe TeX, c'est que les arguments de la macro peuvent être *délimités*, ce qui signifie qu'un ou plusieurs caractères peuvent venir s'intercaler entre eux (et éventuellement en tête et en queue de liste) dans la définition de la macro. Commençons par un exemple simple : une macro à un argument.

```
\def\test#1-{?‘?‘?‘ #1???}
```

(le digramme `?‘` produit le point d'interrogation inversé « ¿ » cher aux hispanophones), où nous avons délimité l'unique argument par un tiret. À l'utilisation de la macro, un tiret analogue devra alors se trouver après l'argument, et celui-ci s'étendra jusqu'au premier tiret rencontré, quel que soit le nombre des caractères qui le composent ; il doit cependant être entendu qu'un tiret « masqué » à l'intérieur d'une paire d'accolades ne compte pas. Voici quelques exemples d'utilisation de la macro « `\test` » ainsi définie :

```
\test123-
```
donne : ¿¿¿ 123??? et

```
\test abc-
```
donne : ¿¿¿ abc???;

```
\test N'est-ce pas pratique-
```
donne : ¿¿¿ N'est???ce pas pratique-, ce qui n'est sans doute pas le but recherché ;

```
\test{N'est-ce pas pratique}-
```
donne : ¿¿¿ N'est-ce pas pratique??? et

`\test N'est{-}ce pas pratique-`
donne également : ¿¿¿ N'est-ce pas pratique ? ? ?

Si le compilateur ne trouve pas le caractère qui délimite l'argument « à distance raisonnable », il s'arrêtera en imprimant le message d'erreur « `Use of \test doesn't match its definition.` ». Pour comprendre la notion de distance raisonnable, il faut se souvenir que les arguments des macros ne peuvent pas, en principe, s'étendre sur plusieurs paragraphes. Si donc le compilateur rencontre une ligne blanche avant le premier tiret, il ne sera pas content et le signalera en se bloquant et en émettant le message « `Paragraph ended before \test was complete.` ».

Les caractères délimitant les arguments d'une macro peuvent être *a priori* quelconques : une lettre, un chiffre, un symbole de ponctuation, et même un blanc ou un « retour chariot » (qui pourra être codé `\par` dans la définition de la macro). Il est donc possible d'avoir un argument qui est tout simplement toute la fin du paragraphe en cours. Si des blancs sont utilisés pour délimiter les arguments, il faut absolument se souvenir de quelques principes de base de la syntaxe TEX : tous les caractères invisibles comptent comme blancs (par exemple les tabulations) ; plusieurs blancs ont le même effet qu'un seul ; les blancs qui suivent une commande sont ignorés.

Un premier exemple pratique

Voici un exemple d'utilisation « grandeur nature » de ces possibilités.

J'ai expliqué dans le n° 11 de votre rubrique (*Losanges* **32**, 59–63) que pour me constituer mes listes d'étudiants, j'allais puiser l'information dans les classeurs (au format *.xls*) que l'Université tient à ma disposition sur son intranet. Comme la feuille principale de ce classeur contient beaucoup de données qui ne me sont pas utiles, j'y crée une seconde feuille sur laquelle je recopie tout, pour ensuite supprimer les colonnes non pertinentes. Il me reste alors, sur trois colonnes, les numéros matricules, les noms et les prénoms (pour simplifier, je laisse tomber, ici, le sexe et le code de la filière d'études). Cela donne quelque chose dont la partie qui nous intéresse ressemble à

	A	B	C
1	20161247	Lecomte	Patrick
2	20161107	Leduc	Léticie
3	20154134	Leprince	Yves
4	20163104	Le Roy	Younès

Rien de plus simple que de sélectionner ces cellules et de les copier pour les coller ensuite dans un document LaTeX ; les changements de colonne du tableur deviennent des tabulations (donc des blancs pour le compilateur — mais non pour l'éditeur de texte !) et les passages d'une ligne à l'autre des retours chariot. Ce document contient donc une suite de lignes du type

`20161247 Lecomte  Patrick`

(où les blancs de différentes largeurs sont en réalité des *tab*). La solution que je proposais dans le n° 11 était d'utiliser l'éditeur de texte pour remplacer tous les « *tab* » par « `}{` » et tous les « *ret* » par « `}`*ret*`\fiche{` », de manière à obenir un document LaTeX formé (uniquement) de lignes comme

`\fiche{20161247}{Lecomte}{Patrick}` ;

ce document était alors appelé par un document principal contenant notamment la définition

`\newcommand{\fiche}[3]{...}`,

où les points de suspension représentent des instructions qui dépendent de la mise en forme que je désire pour ma liste. Cette définition de la macro `\fiche` pourrait aussi bien être donnée « à la TEX » :

`\def\fiche#1#2#3{...}`.

À ce stade, vient à l'idée de faire des différents champs des arguments délimités… par les *tab*, au lieu de remplacer ceux-ci par des paires d'accolades ! Hélas, cette idée ne fonctionne pas, car les *tab*, rappelons-le, sont traités par le compilateur exactement comme des blancs ; dans notre idée, les différents arguments seraient donc délimités par des blancs, et ceci va immanquablement être source de problème si l'un des noms ou l'un des prénoms contient un blanc (dans notre mini-exemple, c'est le cas de Le Roy). Une solution serait d'inclure ces noms et ces prénoms dans des paires d'accolades, mais ceci exige une recherche et des manipulations qui font perdre tout le bénéfice de notre potentielle astuce.

De toute manière, il reste à délimiter le dernier argument et à placer en début de chaque ligne la com-

mande `\fiche`. Une solution efficace est d'effectuer les manipulations nécessaires non pas à l'arrivée, dans le document LaTeX, mais au départ, dans le classeur. Nous allons y ajouter des colonnes, après les trois qui s'y trouvent déjà, et y placer les symboles que nous adopterons comme délimiteurs, par exemple des astérisques ; en outre, dans une colonne ajoutée tout à gauche, nous recopierons le texte « \fiche ». Ces manipulations ne prennent que quelques secondes, même si notre liste comprend plusieurs centaines d'étudiants. À ce stade, le classeur contient, par exemple :

	A	B	C	D	E	F	G
1	\fiche	20161247	*	Lecomte	*	Patrick	*
2	\fiche	20161107	*	Leduc	*	Léticie	*
3	\fiche	20154134	*	Leprince	*	Yves	*
4	\fiche	20163104	*	Le Roy	*	Younès	*

et l'importation de chacune de ses lignes dans le document LaTeX produit par exemple

```
\fiche 20161247 * Lecomte * Patrick *.
```

Il ne nous reste plus qu'à définir notre macro par

```
\def\fiche#1 * #2 * #3 *{...}.
```

Les délimiteurs entre les arguments sont donc formés des trois caractères « *blanc, *, blanc* » — ou « *tab, *, tab* » : pour le compilateur, c'est la même chose.

Un second exemple — en fait, la suite du premier

Il est possible de se simplifier davantage encore l'existence ; c'est fou comme la paresse est source de créativité.

Supposons que les listes d'étudiants que je gère de la sorte soient celles de mon cours de code MATH1789. Je nommerai alors *MATH1789(16-17).tex* le document dans lequel je recopierai les matricules, noms et prénoms exportés du classeur, et *MATH1789(16-17)L.tex*, avec un *L* en plus, le document principal, dont la compilation créera la liste en allant lire l'information dans *MATH1789(16-17).tex*. (Je rappelle que j'utilise aussi d'autres documents principaux, *MATH1789(16-17)F.tex* par exemple, pour disposer de la liste dans différentes présentations.)

Eh bien, ce document principal, si je veux l'adapter à un autre cours ou à une année ultérieure, je n'ai rien à y changer ! Je le clone et je change son nom en *MATH1685(16-17)L.tex* pour un autre cours ou en *MATH1789(17-18)L.tex* l'année suivante, mais je n'ai pas à toucher à son contenu. Par quel miracle cela est-il possible ? C'est tout simplement que toute l'information nécessaire se trouve dans le *nom* du document : il suffit d'aller l'y récupérer.

Regardons comment cela fonctionne.

Le préambule de ce document en est évidemment la partie la plus importante ; outre les habituelles commandes d'insertion des modules utiles et celles qui règlent les questions de mise en page, il contient évidemment la définition de la macro `\fiche`.

Enfin, vienent les commandes

```
\def\un#1(#2-#3)#4{#1}
\def\deux#1(#2-#3)#4{#2}
\def\trois#1(#2-#3)#4{#3}
\def\codecours{\expandafter\un\jobname}
\def\anneeun{\expandafter\deux\jobname}
\def\anneedeux{\expandafter\trois\jobname}
```

Analysons-les. Les macros `\un`, `\deux` et `\trois` sont des macros à quatre arguments délimités, mais seuls les trois premiers importent ; le quatrième sera de toute manière ignoré. Ces commandes, comme leur nom tente de l'indiquer, renvoient respectivement leur premier, leur deuxième et leur troisième argument.

Pour comprendre les trois lignes qui suivent, il faut d'abord savoir que la macro `\jobname` (prédéfinie par TeX) contient le nom du document (sans son extension *.tex*) ; donc, ici, la chaine de caractères `MATH1789(16-17)`.

La macro `\expandafter`, quant à elle, a pour effet que son deuxième argument est développé avant le premier. Par conséquent,

```
\expandafter\un\jobname
```

équivaut à

```
\un MATH1789(16-17);
```

comme `\un` renvoie son premier argument et que celui-ci est délimité par une parenthèse ouvrante, la macro `\codecours` reçoit la valeur `MATH1789`. De la même manière, `\expandafter\deux\jobname` équivaut à `\deux MATH1789(16-17)`, de sorte que `\anneeun` est défini comme `16`, puisque le deuxième argument de `\deux` est tout ce qui est compris entre la parenthèse ouvrante et le tiret. Et, pareillement, la dernière ligne de ce groupe de commandes attribue à `\anneedeux` la valeur `17`. Lors de l'utilisation

des macros `\un`, `\deux` et `\trois`, leur quatrième argument est le caractère qui suit la parentèse fermante, soit `L`, mais peu importe puisque ce quatrième argument n'est pas utilisé.

Le corps du document contient simplement deux choses : les lignes

```
\begin{center}
{\Large\bf\codecours\ ---
20\anneeun--20\anneedeux}\\
(\today)
\end{center}
```

qui produiront le titre suivant :

MATH1789 — 2016–2017
(8 février 2017)

(`\today` est une commande prédéfinie par TEX, qui imprime la date du jour (de la compilation), en anglais par défaut, mais en français si nous avons inclus le module *babel* avec l'option *french*) et la ligne

```
\input{\codecours(\anneeun-\anneedeux)}
```

qui signifie donc `\input{MATH1789(16-17)}` et qui fait lire la liste des étudiants, chacun dans sa « `\fiche` ».

Outre sa facilité intrinsèque, cette manière de procéder nous évite l'erreur, trop souvent vécue, d'oublier de mettre à jour l'année lorsque nous adaptons un document d'une année à l'autre.

Macros « longues »

Ce titre de section est quelque peu impropre ; une expression plus correcte serait *macros à longs arguments*.

Il a été dit plus haut qu'en principe, l'argument d'une macro ne peut pas s'étendre sur plusieurs paragraphes. La raison en est principalement historique : à l'époque ou TEX a été développé, la capacité en mémoire vive des ordinateurs, même des *mainframes*, était beaucoup plus limitée que celle d'un portable d'aujourd'hui. Or, lorsque le compilateur lit une macro, il doit dans un premier temps avaler ses arguments avant de les digérer. Des arguments trop longs risquaient donc de provoquer des blocages gastriques de la machine. Si ce problème a probablement disparu aujourd'hui, il n'en reste pas moins qu'il est de bonne pratique de ne pas faire agir des macros sur de trop longues portions du document source. En LATEX, la notion d'environnement a précisément été introduite pour remplacer des commandes qui auraient dû porter sur, peut-être, plusieurs pages.

Malgré tous ces arguments de bon sens, il peut arriver, dans certaines circonstances exceptionnelles, que l'on doive construire une macro susceptible de digérer plusieurs paragraphes. Avec la syntaxe LATEX (`\newcommand`), c'est impossible. Avec la syntaxe TEX, en revanche, il suffit de faire précéder la commande `\def` de `\long`.

Grosses macros

Le nombre d'arguments d'une macro est donc limité à neuf. J'imagine que c'est pour de bêtes raisons syntaxiques : le compilateur, ainsi, lorsqu'il rencontre un `#` dans le document source, ne doit pas s'inquiéter de savoir s'il est suivi d'un seul chiffre ou de plusieurs.

La question est : y a-t-il moyen de contourner cette limitation ? La réponse est affirmative. L'astuce est de définir une macro qui ne traite que les neuf premiers arguments, puis passe la main à une consœur qui s'occupera des suivants.

Voici un premier exemple. Mathilde rédige un cours dans lequel apparaissent beaucoup de matrices symétriques d'ordre 4 ; aussi voudrait-elle s'épargner de la frappe en utilisant une macro à laquelle il ne faut donner, comme arguments, que les… 10 éléments de la moitié supérieure de cette matrice (dans l'ordre a_{11}, a_{12}, a_{13}, a_{14}, a_{22}, a_{23}, a_{24}, a_{33}, a_{34}, a_{44}). Sa solution utilise les deux macros suivantes :

```
\def\msym#1#2#3#4#5#6#7#8#9%(1)
{\left(\begin{array}{*4r}
#1&#2&#3&#4\\
#2&#5&#6&#7\\
#3&#6&#8&#9\\
#4&#7&#9&\msymfin}

\def\msymfin#1{#1\end{array}\right)}.
```

De la sorte, le code

```
$\msym123456{-7}890$
```

(1) Rappelons que le symbole *pour cent* est le symbole de commentaire : tout ce qui suit, sur sa ligne, est ignoré par le compilateur. Ici, pensez-vous peut-être, il est en fin de ligne et ne masque donc rien. Si, cependant : il masque le « retour chariot », qui équivaut à un blanc ; celui-ci délimiterait le neuvième argument, ce qui n'est pas souhaitable ici.

produit

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & -7 \\ 3 & 6 & 8 & 9 \\ 4 & -7 & 9 & 0 \end{pmatrix}.$$

Cependant, un peu plus loin, Mathilde voudrait faire de même pour des matrices symétriques d'ordre 5… et elle réalise que sa solution pour les matrices d'ordre 4 ne fonctionne que parce que, heureux hasard, la macro n'a plus besoin des neuf premiers arguments au moment de lire le dixième. Et ce n'est plus le cas avec ses matrices d'ordre 5 ! Elle devra donc, ici, utiliser une solution un peu plus lourde, où ceux des premiers arguments qui devront resservir (les quatrième, cinquième, huitième et neuvième) seront stockés :

```
\def\Msym#1#2#3#4#5#6#7#8#9%
{\def\argquat{#4}\def\argcinq{#5}
\def\arghuit{#8}\def\argneuf{#9}
\left(\begin{array}{*5c}
#1&#2&#3&#4&#5\\
#2&#6&#7&#8&#9\\
#3&#7&\Msymfin}

\def\Msymfin#1#2#3#4#5#6%
{#1&#2&#3\\
\argquat&\arghuit&#2&#4&#5\\
\argcinq&\argneuf&#3&#5&#6
\end{array}\right)}.
```

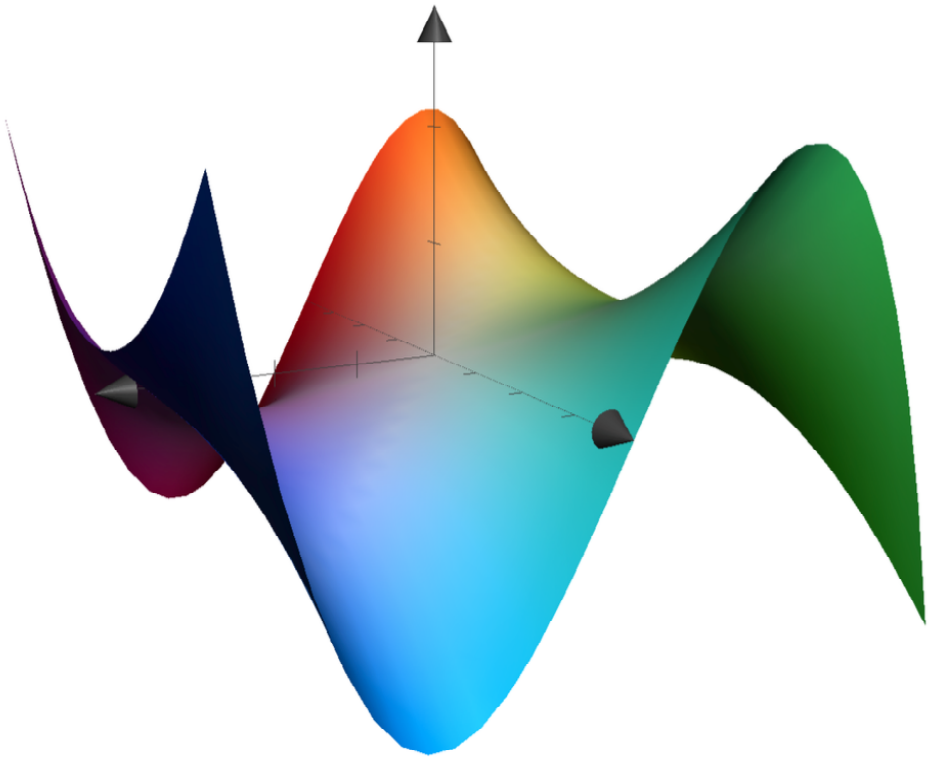
Mathilde n'a plus qu'à coder

```
$\Msym123456789{10}{11}{12}{13}{14}{15}$
```

pour obtenir

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 6 & 7 & 8 & 9 \\ 3 & 7 & 10 & 11 & 12 \\ 4 & 8 & 11 & 13 & 14 \\ 5 & 9 & 12 & 14 & 15 \end{pmatrix}.$$

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



Graphique de $z = (x^3 - 3x)(1 + y^2)$ réalisé avec Grapher © J.-M. Desbonnez