2 Gestion des fontes avec LaTeX 2ε

Si on ne configure rien, par défaut, LaTeX utilise une fonte qui s'appelle Computer Modern, c'est celle que j'utilise dans ce livre (je la trouve lisible, agréable et jolie, mais ça n'engage que moi).

Il en existe bien d'autres : d'autres dessins des fontes latines, bien entendu, mais également des fontes qui présentent d'autres alphabets. Pas seulement le cyrillique ou le grec, auxquels tu dois t'attendre, mais aussi d'autres langues. J'ai réservé les fontes non-latines à la section suivante, sur les langues.

Ce que je présenterai ici, ce sont les différents packages permettant de changer de fonte, pas ce qui te permet de traiter telle ou telle langue. Ainsi, la façon de passer du français au russe se trouvera à la section suivante (section 3 page 166), par contre les packages proposant plusieurs fontes pour le russe se trouveraient ici, si j'en avais trouvé plusieurs.

Je te présenterai également ici quelques packages qui se trouvent à mi-chemin entre le dessin et les fontes, par exemple pour dessiner des touches de clavier, pour souligner différemment, pour faire du braille, etc.

2.1 Pandora : le package `pandora`

Ce package de Frank Mittelbach [142] permet, pour sa part, de sélectionner les fontes « Pandora » à la place des « Computer Modern » usuelles. Je les trouve un peu moins jolies, mais c'est chacun selon son goût.

Je t'invite cordialement à lire le fichier `pandora.sty` pour avoir un aperçu intéressant de NFSS. La suite en « Pandora » [2].

---

[2] Si tu veux utiliser cette fonte dans tout ton document, c'est facile, tu utilises le package et c'est bouclé. Par contre, si tu veux, comme moi, l'utiliser seulement pour un passage (par exemple pour faire des encadrés), ce qui est *plutôt une mauvaise idée*, il faut ruser un peu. En fait, quand LaTeX cherche à faire du texte en Romain (texte droit normal) il prend la famille de fonte définie dans `\rmdefault`. Par défaut, cette commande est définie comme `cmr`, pour obtenir du Computer Modern Romain. Pour obtenir du Pandora, il suffit de re-définir cette commande comme ça :

```
\renewcommand\rmdefault{panr}
```

Il en va de même pour la famille Sans-serif, la définition par défaut de `\sfdefault` renvoie vers `cmss`, il suffit de redéfinir cette commande pour qu'elle renvoit vers `pss` :

```
\renewcommand\sfdefault{pss}
```

Une autre particularité est que la commande `\textbf` renvoit sur `\bfdefault` qui vaut en temps normal `bx` (Bold Extended). La fonte « Pandora », pour sa part, ne prévoit que du gras, pas « extended », du coup il faut re-définir `\bfdefault` pour que ça renvoit vers `b` :

```
\renewcommand\bfdefault{b}
```

De même, Pandora ne prévoit pas d'italique, mais seulement du penché, alors soit tu t'abstiens d'utiliser `\textit` et tu utilises `\textsl`, soit tu fais en sorte que l'italique renvoit vers du penché :

```
\renewcommand\itdefault{sl}
```

Enfin, la fonte « Pandora » n'existe que dans le schéma de codage `OT1`, il faudra donc utiliser ce schéma de codage (la conséquence immédiate est que les mots contenant des caractères accentués ne pourront plus être coupés après le premier accent) :

```
\renewcommand\encodingdefault{OT1}
```

Après ces modifications, il faudra ajouter un appel à `\rmfamily`, pour faire jouer l'algorithme de choix de fonte de LaTeX, et que le basculement soit effectif.

Voici un exemple d'utilisation de la fonte Pandora. C'est une fonte classique dans le monde TEX (ça fait très longtemps qu'elle est diffusée quasi-systématiquement avec TEX et LATEX). Cette fonte est nettement plus noire que la fonte Computer Modern habituelle de LATEX.

Elle est disponible en Romain (texte normal), en *penché (pas d'italique, l'italique est en fait du penché)*, et en **gras**. Par contre, elle n'est pas disponible en petites capitales. La variété Sans-Serif existe, en normal, en **gras** et en *penché*. Les variétés penchées et grasses n'existent pas (ou au moins ne sont pas accessibles via le package).

Je n'aime pas particulièrement cette fonte, mais je te la montre, au cas où elle te tenterait.

La suite en Computer Modern, bien entendu.

2.2 Fonte concrete : le package `beton`

Le package `beton`, écrit par Frank Jensen [93] permet d'utiliser les fontes « Concrete[3] » de la même façon que le package `pandora` permettait d'utiliser les fontes « Pandora ».

Ce paragraphe est en « Concrete ». Tout, ou presque, est faisable puisque cette fonte est construite sur la base des Computer Modern. Le package `beton` ne prévoit que la famille du Romain (et donc pas la famille du sans-serif ni la famille du télétype). La famille du Romain est disponible en normal (bien entendu, c'est ce que tu es en train de lire), en *italique*, en PETITES CAPITALES, en *penché*, en **gras**[4], en ***italique grasse***, en ***penché gras***, mais pas en petites capitales grasses.

Cette fonte propose également une variété **« semi bold condenced », c'est-à-dire semi gras condencé (comme ça)**.

Pour revenir à du Computer Modern, il faut redéfinir tout ça pour revenir à la normale :

```
\renewcommand\encodingdefault{T1}
\renewcommand\rmdefault{cmr}
\renewcommand\sfdefault{cmss}
\renewcommand\bfdefault{bx}
\renewcommand\itdefault{it}
\fontencoding{T1}\selectfont
```

[3]Cette fonte existe dans les deux codages classiques de LATEX, `OT1` et `T1`. La famille correspondant au Romain est `ccr`, pour utiliser cette fonte sans changer tout le document (comme je le fais ici), il faut renvoyer LATEX vers cette famille :

```
\renewcommand\rmdefault{ccr}
```

Le fonte « Concrete » fonctionne à peu près comme « Computer Modern » (c'en est en fait une version modifiée). Toutefois elle prévoit une version « semi bold condensed » qui n'existe pas en temps normal. Pour déclarer cette série (c'est une série, au même titre que le gras), il faut ajouter les lignes suivantes dans l'en-tête de ton document :

```
\newcommand\sbcdefault{sbc}
\DeclareRobustCommand\sbcseries{\fontseries\mcdefault\selectfont}
\DeclareTextFontCommand{\textsbc}{\sbcseries}
```

Les deux commandes `\sbcseries` et `\textsbc` existent alors. Par contre, comme très peu de fontes prévoient cette variété « semi bold condensed », il y a des chances que ces commandes ne marchent que pour cette fonte. Une solution pour que la commande marche partout sans produire d'erreur est de lui dire qu'en temps normal `\sbcdefault` vaut `bx`, ça indique d'utiliser le « Bold Extended » à la place du `sbc`. Bien entendu, dans ce cas-là, lors du passage à « Concrete », il faut indiquer que `\sbcdefault` est `sbc`.

Pour revenir en « Computer Modern », il suffit de redéfinir la famille du romain et de demander un changement de fonte :

```
\renewcommand\rmdefault{cmr}
\rmfamily
```

[4]Remarque que le gras n'existe pas pour les fontes « Concrete », mais les fichiers de configuration indiquent d'utiliser le gras habituel de « Computer Modern ». Les deux fontes étant issues de la même souche, le résultat est impeccable.

2.3 Fonte concrete : le package `concmath`

Ce package, que nous devons à Ulrik VIETH [215], permet, comme le package `beton`, d'utiliser la fonte Concrete, mais également en mode maths.

On se contentera de quelques exemples de ce que peut produire ce package. Pour pouvoir comparer, un exemple de mathématiques « normales », *i.e.* en Computer Modern :

$$\sum_{i=0}^{n} u_i = \ell \in \mathbb{R}$$

En concrete, ça donnera le résultat suivant, légèrement plus sombre, et plus adapté a un texte en « Concrete[5]. »

$$\sum_{i=0}^{n} u_i = \ell \in \mathbb{R}$$

Et la version grasse en « Concrete » ressemble à ça (elle n'existe pas, mais le package fait en sorte de la remplacer par le gras de Computer Modern, qui d'adapte bien puisque ces deux fontes sont de la même famille, ce que tu vois ici est donc, en fait, du gras Computer Modern) :

$$\sum_{\boldsymbol{i=0}}^{\boldsymbol{n}} \boldsymbol{u_i} = \boldsymbol{\ell} \in \mathbb{R}$$

Ce package admet quatre options. La première, exscale, fait appel au package `exscale`, et le rend compatible avec la fonte « Concrete ». La deuxième, amsfonts, fait appel au package `amsfonts`, et redéfinit les fontes de symboles pour utiliser du « Concrete » là où c'est possible. La troisième, amssymb, fait la même chose avec le package `amssymb`. Enfin, la quatrième option, sansbold, utilise du Computer Moderne sans-serif semi-gras pour les textes en gras au lieu du gras Computer Modern (en effet, la fonte Concrete n'existe pas en gras).

---

[5]Si tu veux que toutes tes équations utilisent la fonte « Concrete », par exemple quand tu fais tout un document en « Concrete », il suffit d'utiliser le package. Par contre, si tu veux, comme moi, ne produire que certaines équations en « Concrete », alors il va falloir ruser un peu plus. En effet, le package écrase les définitions des fontes mathématiques de LaTeX, sans prévoir de possibilité de retour arrière. L'idée est de créer une nouvelle version des maths (comme `bold` est une version, qu'on appelle avec `\mathversion`), et de spécifier que dans cette nouvelle version on va utiliser la fonte voulue.

D'abord créer cette nouvelle version :

```
\DeclareMathVersion{concrete}
```

Ensuite, indiquer les fontes à utiliser pour cette version (toutes les fontes qu'on ne précisera pas seront prises dans la version `normal`, ce qui fait que les symboles évolués des divers packages, qui n'existent souvent qu'en une seule version, seront quand même disponibles, mais n'auront pas été adaptés) :

```
\SetSymbolFont{operators}{concrete}{OT1}{ccr}{m}{n}
\SetSymbolFont{letters}{concrete}{OML}{ccm}{m}{it}
\SetSymbolFont{symbols}{concrete}{OMS}{ccsy}{m}{n}
\SetSymbolFont{largesymbols}{concrete}{OMX}{ccex}{m}{n}
```

Si tu veux également la version en gras, même chose, on définit une nouvelle version, `concretebold` :

```
\DeclareMathVersion{concretebold}
\SetSymbolFont{operators}{concretebold}{OT1}{ccr}{bx}{n}
\SetSymbolFont{letters}{concretebold}{OML}{ccm}{b}{it}
\SetSymbolFont{symbols}{concretebold}{OMS}{ccmsy}{b}{n}
```

À noter que `largesymbols` n'existe pas en Concrete gras.

Certaines des fontes du package `amsmath` existent également en variété Concrete. Si tu veux qu'elles soient accessibles dans tes deux nouvelles version des maths, voilà comment faire (si tu as chargé `amsmath` avant, bien entendu) :

```
\SetSymbolFont{AMSa}{concrete}{U}{cca}{m}{n}
\SetSymbolFont{AMSb}{concrete}{U}{ccb}{m}{n}
```

Si tu veux utiliser ces symboles en gras, il te faudra reprendre ces deux commandes pour la version `concretebold` : ces symboles n'existent pas en gras, il suffit donc d'aller chercher les versions non grasses.

2.4 Fonte concrete : le package `ccfonts`

Ce petit package, que nous devons à Walter SCHMIDT [190], regroupe les fonctionnalités de `beton` et de `concmath`, à savoir : passer en fonte Concrete tant pour le texte que pour les maths.

Il peut prendre trois options utiles.

La première, exscale, appelle le package `exscale` et fait en sorte qu'il soit compatible avec la fonte Concrete.

La deuxième, slantedGreek, indique que l'on souhaite avoir des lettres grecques majuscules penchées (en italique), comme on en a l'habitude dans les équations en Computer Modern. En effet, en Concrete, par défaut, ces lettres sont droites.

La troisisème, boldsans, indique que l'on souhaite utiliser du Computer Modern sans-serif semi-gras en guise de gras. Par défaut, on utilisera le gras normal de Computer Modern.

2.5 Times et ses associés : le package `txfonts`

Ce package se chargera de basculer *tout ton document* dans des fontes plus connues en dehors du monde TeX : Times, et les fontes associées[6].

Ce petit texte tout simple pour te donner un exemple : c'est curieux comme ce simple changement de fonte donne l'impression de ne plus lire du TeX. Essaye sur un de tes documents : tu verras, c'est stupéfiant. Bien entendu, ça marche dans toutes les variantes classiques : **en gras**, en sans-serif, en *italique*, EN PETITES CAPITALES, `en télétype`, *en penché* ; et même en faisant des mélanges : **sans-serif gras**, ***italique grasse***, **PETITES CAPITALES GRASSES**, ***penché gras***, *sans-serif italique*, PETITES CAPITALES TÉLÉTYPE, etc.

Ce package change également les fontes utilisées dans les équations. Pour rappel, en Computer Modern, une équation ressemblera à ça :

$$\Delta \leqslant \widetilde{abcd} \simeq \sum_{i=0}^{n} u_i \neq \int_0^x f(t)dt \in \mathbb{R}$$

[6]Si tu veux utiliser ce que fait ce package sans pour autant changer tout ton document, voilà les redéclarations utiles :

```
\renewcommand\rmdefault{txr}
\renewcommand\sfdefault{txss}
\renewcommand\ttdefault{txtt}
\rmfamily
```

Pour revenir au Computer Modern :

```
\renewcommand\rmdefault{cmr}
\renewcommand\sfdefault{cmss}
\renewcommand\ttdefault{cmtt}
\rmfamily
```

En utilisant les fontes Times du package `txfonts`, le résultat sera[7] :

$$\Delta \leqslant \widetilde{abcd} \simeq \sum_{i=0}^{n} u_i \neq \int_0^x f(t)dt \in \mathbb{R}$$

On pourra même utiliser la variante bold des maths, pour obtenir par exemple (en Computer Mordern) :

$$\boldsymbol{\Delta \leqslant \widetilde{abcd} \simeq \sum_{i=0}^{n} u_i \neq \int_0^x f(t)dt \in \mathbb{R}}$$

Ou, avec le package :

$$\boldsymbol{\Delta \leqslant \widetilde{abcd} \simeq \sum_{i=0}^{n} u_i \neq \int_0^x f(t)dt \in \mathbb{R}}$$

2.6 Palatino : le package `palatino`

Ce package, que nous devons à Sebastian RAHTZ, et livré en standard avec LaTeX, permet d'utiliser les fontes « Palatino » si elles sont installées sur ton ordinateur (c'est normalement le cas pour la majorité des installation modernes)[8]. Ces fontes, dessinées par Herman ZAPF, sont réputées être très lisibles, et sont assez esthétiques.

Ce texte est en Palatino. La fonte est disponible en *italique*, en *penché*, en PETITES CAPITALES, en **gras**, en ***italique gras***, en ***penché gras***, et en **PETITES CAPITALES GRASSES**.

Pour les autres familles habituelles, le package choisit du Courier à la place du télétype (`ça ressemble à ça, c'est ignoblement laid`). C'est disponible en *`penché (pas d'italique)`*, en `PETITES CAPITALES`, en **`gras`**, en ***`gras penché`***, et en **`PETITES CAPITALES GRASSES`**.

[7] Là encore, tu peux faire le changement pour une partie seulement de ton document, en déclarant une nouvelle version, `times`, et pour le gras encore une autre version, `timesbold`. Ça pourra ressembler à ça :

```
\DeclareMathVersion{times}
\SetSymbolFont{operators}{times}{OT1}{txr}{m}{n}
\SetSymbolFont{letters}{times}{OML}{txmi}{m}{it}
\SetSymbolFont{symbols}{times}{OMS}{txsy}{m}{n}
\SetSymbolFont{largesymbols}{times}{OMX}{txex}{m}{n}
\DeclareMathVersion{timesbold}
\SetSymbolFont{operators}{timesbold}{OT1}{txr}{bx}{n}
\SetSymbolFont{letters}{timesbold}{OML}{txmi}{bx}{it}
\SetSymbolFont{symbols}{timesbold}{OMS}{txsy}{bx}{n}
\SetSymbolFont{largesymbols}{timesbold}{OMX}{txex}{bx}{n}
```

Si tu utilises les fontes du package `amsmath`, tu peux ajouter :

```
\SetSymbolFont{AMSa}{times}{U}{txsya}{m}{n}
\SetSymbolFont{AMSb}{times}{U}{txsyb}{m}{n}
\SetSymbolFont{AMSa}{timesbold}{U}{txsya}{bx}{n}
\SetSymbolFont{AMSb}{timesbold}{U}{txsyb}{bx}{n}
```

Le package `txfonts` redéfinit d'autres symboles pour coller avec du Times, mais ce comportement est délicat à obtenir sans charger le package.

[8] Si tu veux utiliser ces fontes pour un bout de texte, mais pas pour tout ton document, voilà comment tu peux procéder :

```
\renewcommand\rmdefault{ppl}
\renewcommand\sfdefault{phv}
\renewcommand\ttdefault{pcr}
\rmfamily
```

Pour revenir au Computer Modern :

```
\renewcommand\rmdefault{cmr}
\renewcommand\sfdefault{cmss}
\renewcommand\ttdefault{cmtt}
\rmfamily
```

Il choisit de l'Helvetica pour le sans-serif (là aussi, le résultat n'est pas terrible, je n'aime pas la fonte Helvetica). Cependant cette fonte à le bon goût d'exister dans un nombre impressionnant de variantes : c'est disponible en *penché (pas d'italique)*, en PETITES CAPITALES, en condensé, en *condensé penché*, en PETITES CAPITALES CONDENSÉES, en **gras**, en ***gras penché***, en **PETITES CAPITALES GRASSES**, en **gras condensé**, en ***penché gras condensé***, et en **PETITES CAPITALES GRASSES CONDENSÉES**.

Les deux séries condensé et condensé gras ne sont pas prévues par le noyau LATEX de base, et ne le sont pas non plus par le package. Pour les obtenir, il te faudra indiquer les commandes suivantes dans ton préambule[9] :

```
\def\mcdefault{m}
\DeclareRobustCommand\mcseries{%
  \fontseries\mcdefault\selectfont
}
\DeclareTextFontCommand{\textmc}{\mcseries}
\def\bcdefault{bx}
\DeclareRobustCommand\bcseries{%
  \fontseries\bcdefault\selectfont
}
\DeclareTextFontCommand{\textbc}{\bcseries}
```

Ça déclare les commandes `\mcseries` et `\textmc` pour obtenir du condensé et `\bcseries` et `\textbc` pour du gras condensé[10].

Si tu préfères utiliser du télétype habituel de LATEX (celui de Computer Modern) quand tu utilises la commande `\texttt`, alors il te faudra ajouter cette commande magique dans ton document :

```
\renewcommand\ttdefault{cmtt}
```

C'est ce que j'ai fait juste avant ce paragraphe, et, franchement, le résultat est nettement moins moche que Courier.

2.7 Encore du palatino : le package `mathpple`

Ce package, que nous devons à Walter SCHMIDT, et également distribué en standard avec LATEX, utilise des fontes « Palatino » pour faire les maths dans un document. Ça se marie, évidemment, très bien avec le package `palatino`.

Du coup, je passe le texte en Palatino. Pour le moment, seul le texte a été modifié. Pour pouvoir comparer, voilà une équation en Computer Modern :

$$\Delta \leqslant \widetilde{abcd} \simeq \sum_{i=0}^{n} u_i \neq \int_0^x f(t)dt \in \mathbb{R}$$

[9] Je viens d'indiquer à LATEX de revenir à sa définition par défaut du télétype, sans quoi les quelques lignes qui suivent auraient été en fonte Courier.

[10] Telles qu'elles viennent d'être déclarées, ces commandes produisent du texte normal, pour `\textmc` et du texte gras pour `\textbc` ; du coup si tu t'en sers dans du texte normal, en Computer Modern, elles ne poseront pas de problème. Pour qu'elles accèdent aux variétés condensées, il suffit d'indiquer :

```
\renewcommand\mcdefault{mc}
\renewcommand\bcdefault{bc}
```

Et maintenant, la même, si tu utilises le package `mathpple`[11] :

$$\Delta \leqslant \widetilde{abcd} \simeq \sum_{i=0}^{n} u_i \neq \int_0^x f(t)dt \in \mathbb{R}$$

Il faut noter que tous les symboles mathématiques ne sont pas modifiés, par exemple, ceux fournis par le package `amsmath` (comme le $\mathbb{R}$) restent les mêmes, mais les lettres, les symboles de grande taille, etc., sont modifiés.

Comme tu le sais, on peut aussi faire des maths en gras (utile pour les titres, par exemple). En Computer Modern, ça donne ça :

$$\boldsymbol{\Delta \leqslant \widetilde{abcd} \simeq \sum_{i=0}^{n} u_i \neq \int_0^x f(t)dt} \in \mathbb{R}$$

Et, maintenant, la même, toujours en gras, mais avec le package `mathpple` :

$$\boldsymbol{\Delta \leqslant \widetilde{abcd} \simeq \sum_{i=0}^{n} u_i \neq \int_0^x f(t)dt} \in \mathbb{R}$$

2.8 Encore du Palatino : le package `mathpazo`

Ce package, que nous devons à Walter SCHMIDT et Diego PUGA, lui aussi distribué avec LaTeX, propose une autre variété de fontes pour les mathématiques avec du Palatino.

Du coup, je repasse le texte en Palatino. Voilà une équation en Computer Modern :

$$\Delta\mathit{\Delta} \leqslant \widetilde{abcd} \simeq \sum_{i=0}^{n} u_i \neq \int_0^x f(t)dt \in \mathbb{R}$$

Ici, la même avec le package `mathpazo` :

$$\blacksquare\mathit{\Delta} \leqslant \widetilde{abcd} \simeq \sum_{i=0}^{n} u_i \neq \int_0^x f(t)dt \in \mathbb{R}$$

De même qu'on peut obtenir la version `bold` avec Computer Modern :

$$\boldsymbol{\Delta\mathit{\Delta} \leqslant \widetilde{abcd} \simeq \sum_{i=0}^{n} u_i \neq \int_0^x f(t)dt} \in \mathbb{R}$$

---

[11] Ici encore, si tu ne souhaites pas modifier toutes tes équations, mais, comme moi, seulement certaines, il faudra déclarer une nouvelle version des mathématiques, `palatino`, et une autre pour le gras, `palatinobold`. Ça peut se faire comme ça :

```
\DeclareMathVersion{palatino}
\SetSymbolFont{operators}{palatino}{OT1}{zpple}{m}{n}
\SetSymbolFont{letters}{palatino}{OML}{zpple}{m}{it}
\SetSymbolFont{symbols}{palatino}{OMS}{zpple}{m}{n}
\SetSymbolFont{largesymbols}{palatino}{OMX}{zpple}{m}{n}
\DeclareMathVersion{palatinobold}
\SetSymbolFont{operators}{palatinobold}{OT1}{zpple}{b}{n}
\SetSymbolFont{letters}{palatinobold}{OML}{zpple}{b}{it}
\SetSymbolFont{symbols}{palatinobold}{OMS}{zpple}{b}{n}
\SetSymbolFont{largesymbols}{palatinobold}{OMX}{zpple}{m}{n}
```

Il suffira ensuite de changer de version avant et après ton équation :

```
\mathversion{palatino}
\[...\]
\mathversion{normal}
```

On peut obtenir une version `bold` avec le package `mathpazo` :

$$\blacksquare\Delta \leqslant \widetilde{abcd} \simeq \sum_{i=0}^{n} u_i \neq \int_0^x f(t)dt \in \mathbb{R}$$

2.9 Avant Garde : le package `avant`

Ce package, que nous devons à Sebastian RAHTZ, qui est livré en standard avec LATEX, charge la fonte « Avant Garde » à la place du sans serif par défaut de LATEX.

Ça ressemble à ça, Avant Garde[12], et ça existe en PETITES CAPITALES, en *penché (pas d'italique)*, en **gras**, en **PETITES CAPITALES GRASSES**, en ***penché gras***.

Ce package ne change que la famille des sans serif, et ne touche pas au reste. Tu peux le combiner avec d'autres packages, bien entendu.

Il faut cependant faire attention à deux points. D'une part cette fonte est plus grosse que le Computer Modern : si tu regardes, le texte en « Avant Garde » à l'air d'être écrit plus gros, alors que je n'ai pas changé de taille. Pour que cette fonte ait l'air de bien se marier avec du texte en 10 points, il vaut mieux choisir un corps 8, comme je le fais ici, par exemple. Pour ce faire tu peux utiliser la commande `\fontsize` qui prend deux paramètres : le premier est la taille voulue, et le deuxième la taille de l'interligne à appliquer. Dans ce cas précis, j'ai choisi un corps 8 et un interligne de 12, parce que c'est le même interligne que le texte romain qui l'entoure. Ça se demande comme ça :

```
\fontsize{8}{12}\selectfont\textsf{comme...
```

Un autre point est que, si tu veux écrire un texte un peu long avec cette fonte (un peu plus qu'un titre, par exemple un paragraphe entier, ou même tout un document), il faut que tu utilises un interligne généreux. En Computer Mordern, on considère qu'il faut un rapport de 1,2 entre le corps de la fonte et l'interligne. Ça donne un interligne de 12 points pour un texte en corps 10. Pour la fonte « Avant Garde », il faut être plus généreux, il vaut mieux viser 1,4 ou 1,5. Ça donne un interligne de 13 points pour un texte en corps 9, par exemple.

2.10 New Century Schoolbook : le package `newcent`

Ce package, que nous devons à Sebastian RAHTZ, distribué de base avec LATEX, permet d'accéder aux fontes « New Century Schoolbook » si elles sont installées sur ton système (c'est en général le cas).

Du « New Century Schoolbook », ça ressemble à ça[13]. C'est une fonte un peu plus noire que le « Computer Modern » que j'utilise dans le reste du livre. Cette fonte est

[12] Si tu veux utiliser cette fonte temporairement dans un document (et non pas systématiquement à la place du sans serif de LATEX), tu peux faire comme ça :

```
\renewcommand\sfdefault{pag}
\rmfamily
```

À partir de là, le sans serif dans ton document sera obtenu avec « Avant Garde ». Pour revenir au sans serif par défaut de LATEX (Computer Modern), il faut faire :

```
\renewcommand\sfdefault{cmss}
\rmfamily
```

[13] Encore une fois, si tu ne souhaites pas utiliser cette fonte tout au long de ton document, mais seulement à un endroit, voilà les déclarations à utiliser :

```
\renewcommand\rmdefault{pnc}
\renewcommand\sfdefault{pag}
\renewcommand\ttdefault{pcr}
\rmfamily
```

Et pour revenir au Computer Modern :

```
\renewcommand\rmdefault{cmr}
```

disponible en *italique*, en PETITES CAPITALES, en *penché*, en ***italique grasse***, en **PETITES CAPITALES GRASSES** et en ***penché gras***.

Le package remplace les variantes Sans-serif par la fonte « Avant Garde[14] ». Cette fonte est disponible en normal (sans-serif), en *penché (pas d'italique)*, en PETITES CAPITALES, en **gras**, en ***penché gras***, et en **PETITES CAPITALES GRASSES**.

Le package remplace également le télétype par du `Courier`. C'est moche, je l'ai déjà dit.

Le package *ne prévoit pas* de modifications sur les mathématiques. Il semble qu'il n'y ait pas, dans les fontes mathématiques classiques, une fonte qui soit prévue pour donner un bon résultat avec du « New Century Schoolbook ». Tu peux essayer de reprendre les fontes pour le Times, le résultat devrait être convenable.

2.11 Bookman : le package `bookman`

Ce package, que nous devons à Sebastian RAHTZ, lui aussi fourni avec LaTeX en standard, permet d'accéder à la fonte « Bookman ». Il n'utilise cette fonte que pour la famille du romain.

Du « Bookman », ça ressemble à ça[15]. Pour cette fonte, le package fait le choix d'utiliser la fonte « Avant Garde » pour la famille du sans-serif et la fonte « Courier » pour la famille du télétype. La fonte « Avant Garde » est décrite au paragraphe 2.10 page précédente.

Pour ce qui est du Bookman, il est disponible en PETITES CAPITALES, en *italique*, en *penché*, en **gras**, en **PETITES CAPITALES GRASSES**, en ***penché gras*** et en ***italique grasse***.

Il n'y a pas de package proposant des maths pour aller avec Bookman.

2.12 Zapf Chancery : le package `chancery`

Cette fonte, dessinée par Herman ZAPF, est relativement célèbre, même s'il vaut mieux ne pas rédiger un livre avec (c'est plutôt destiné à faire un titre, ou une carte de vœux). Le package `chancery`, que nous devons à Sebastian RAHTZ, et qui est distribué en standard avec LaTeX, charge cette fonte à la place du romain de LaTeX.

Du « Zapf Chancery », ça ressemble à ça. Cette fonte n'est disponible que dans une seule famille, une seule série, et une seule forme (pas d'italique, pas de petites capitales, etc).

Si tu veux utiliser cette fonte de manière plus occasionnelle, par exemple via une commande `\textzc`*, voilà comment tu peux le faire :*

```
\def\zcdefault{zc}
\DeclareRobustCommand\zcfamily{%
  \fontfamily\zcdefault\selectfont
}
\DeclareTextFontCommand{\textzc}{\zcseries}
```

```
\renewcommand\sfdefault{cmss}
\renewcommand\ttdefault{cmtt}
\rmfamily
```

[14] Une fois encore, cette fonte est trop grosse par rapport au contexte.

[15] Si tu veux utiliser cette fonte pour un passage seulement de ton document, tu peux le faire en utilisant :

```
\renewcommand\rmdefault{pbk}
\rmfamily
```

Et, bien entendu, pour revenir au Computer Modern :

```
\renewcommand\rmdefault{cmr}
\rmfamily
```

2.13 Charter : le package `charter`

Ce package, que nous devons à Peter DYBALLA, charge la fonte « Charter » à la place du romain de LATEX.

Du « Charter », ça ressemble à ça[16]. Cette fonte existe en PETITES CAPITALES, en *penché*, en *italique*, en **gras**, en **PETITES CAPITALES GRASSES**, en ***penché gras***, et en ***italique grasse***.

Le package ne prévoit pas de variété spécifiquement adaptée pour les familles des sans serif ou du télétype. Tu pourras par exemple utiliser le package `avant` pour utiliser la fonte « Avant Garde » pour le sans serif, ou le package `courier` pour utiliser la fonte « Courier » pour le télétype.

2.14 KNUTH n'avait pas voulu ça : le package `cmsd`

Dans l'ancien temps, dans les années 80, TEX utilisait les fontes dites « Computer Modern » telles qu'elles avaient été écrites par Donald E. KNUTH.

En ce temps-là, quand on écrivait très gros, en Sans-Serif, et en gras, ça ressemblait à ça :

Essai gras.

Pour pouvoir bénéficier de la césure dans les langues accentuées, il a fallu se mettre à utiliser des fontes codant tous les caractères[17] accentués.

Ce sont ces fontes que LATEX utilise quand on fait appel au schéma de codage T1 (option T1 du package `fontenc`).

Il existe deux variantes, dans « EC », du Sans-serif-gras. L'une pourrait être qualifiée de semi-grasse, c'est celle utilisée par LATEX quand on fait appel à `\textsf` et `\textbf` en même temps. L'autre, qui pourrait être qualifiée de grasse, n'est pas utilisée par LATEX par défaut. La différence ne se voit véritablement que sur les grandes tailles.

Du coup, de nos jours, dans un document en codage T1, comme ce livre, lorsqu'on demande du Sans-serif-gras en grand taille, on obtient ça :

Essai gras.

ce qui n'a pas grand chose à voir avec ce qu'on obtenait avec les fontes de KNUTH.

Le petit package `cmsd`, que nous devons à Walter SCHMIDT [189], change discrètement la définition du Sans-serif de LATEX pour utiliser la variante qui ressemble plus à ce que proposait KNUTH, à savoir :

[16] Si tu veux n'utiliser cette fonte que sur une partie de ton document, tu peux faire :

```
\renewcommand\rmdefault{bch}
\rmfamily
```

Pour revenir à « Computer Modern », comme d'habitude :

```
\renewcommand\rmdefault{cmr}
\rmfamily
```

[17] TEX ne sait faire de césure que dans des mots, or pour lui un mot est une suite de lettres (ça semble raisonnable), or dans les fontes « Computer Modern » originales, pour faire une lettre accentuée, on collait un caractère accent sur un caractère lettre. TEX ne considère pas une telle construction comme une lettre, et donc refuse de césurer un mot au-delà d'une construction pareille. En utilisant des fontes dans lesquelles les lettres accentuées étaient directement disponibles, on pouvait résoudre ce problème. Ces fontes, qui sont directement reprises de celles de KNUTH, mais avec un codage comprenant les lettres accentuées, sont les fontes dites « EC » que l'on utilise de nos jours dans tous les documents LATEX.

Essai gras.

Et, il faut reconnaître que, là, on voit bien que c'est en gras !

2.15 Le package `oldstyle`

Ce package, que nous devons à Robin FAIRBAIRNS [63] permet de produire des nombres comme en utilisaient les anciens typographes, tu sais, des trucs du genre 123456. C'est assez joli.

Pour ce faire, ce package crée trois commandes. La plus utile, qui est celle que je viens d'utiliser à l'instant est `\textos` qui donne accès à ces chiffres. Son pendant direct, qui permet la même chose, mais avec une syntaxe plus proche de LaTeX 2.09 est `\oldstyle`. Elle s'utilise à l'ancienne manière, c'est-à-dire qu'au lieu d'appeler

```
\textos{123456}
```

on appelle

```
{\oldstyle 123456}
```

Enfin, la troisième commande est `\mathos` qui permet d'utiliser ces chiffres dans une équation. C'est un alphabet mathématique comme un autre.

Bien qu'il soit possible de s'en servir en mathématiques, via la commande `\mathos`, il vaut mieux réserver cette forme aux chiffres que tu utilises dans un contexte plus textuel, par exemple pour numéroter les équations ou les tableaux, ou encore pour les dates. Ça semble plutôt une mauvaise idée de s'en servir pour désigner des quantités mathématiques.

2.16 Petites capitales : le package `slantsc`

Ce petit package, que nous devons à Harald HARDERS [75], modifie la façon dont LaTeX manipule ses fontes. Le but est de pouvoir obtenir des petites capitales penchées.

Dans son mode de fonctionnement, LaTeX choisit la fonte en cours en fonction de 5 paramètres (voir à ce sujet le paragraphe 7.3 page 34, ou pour plus de détails, le paragraphe 11 page 607 au chapitre neuvième). L'un de ces cinq critères est la *forme*, qui peut être italique, petites capitales, droite, ou penchée. Comme chaque critère ne peut avoir qu'une seule valeur à un moment, ce modèle interdit que l'on ait des petites capitales penchées.

La solution proposée par le package est assez simple : il crée d'autres formes. Ces formes sont « petites capitales penchées », et « petites capitales italiques ». Et il va modifier toutes les commandes de changement de forme. Chacune va faire plus que son changement naturel :

- `\upshape`, qui repasse en droit, va en fait passer en « petites capitales » (`sc`) si on était en « petites capitales italiques » (`scit`) ou en « petites capitales penchées » (`scsl`), ou en « droit » (`up`) sinon.
- `\itshape`, qui repasse en italique, va en fait passer en `scit` si on était en `sc` ou en `scsl`, et va passer en « italique » (`it`) sinon.
- `\slshape`, qui repasse en penché, va en fait passer en `scsl` si on était en `sc` ou en `scit`, et va passer en « penché » (`sl`) sinon.
- `\scshape`, qui repasse en petites capitales, va en fait passer en `scsl` si on était en `sl` ou en `scsl`, en `scit` si on était en `it` ou `scit` ; en `sc` sinon.

Du coup, tu peux continuer à utiliser tes commandes habituelles, à cette différence près que maintenant, tu pourras combiner penché et petites capitales (les petites capitales italiques n'existent pas et sont automatiquement remplacées par du penché).

On peut donc obtenir des *PETITES CAPITALES PENCHÉES*, mais aussi des ***PETITES CAPITALES PENCHÉES GRASSES***.

La seule limitation est que ce package ne fonctionne qu'avec les fontes Computer Modern, parce que la majorité des autres fontes ne prévoient pas autant de combinaisons.

2.17 Petites capitales : le package `smallcap`

Nous devons ce package à Daniel TAUPIN [203]. Il vise à règler le même problème que le package `slantsc` : obtenir des petites capitales penchées. Il a une immense qualité : il ne modifie pas la mécanique de gestion des fontes de LATEX. Il a un immense défaut : il perd les `PETITES CAPITALES TÉLÉTYPE` dans la bataille.

Il s'utilise de manière très naturelle, en combinant les commandes habituelles, pour obtenir des *PETITES CAPITALES PENCHÉES*, et des ***PETITES CAPITALES GRASSES PENCHÉES*** (la variante petites capitales italiques grasses n'existe pas et pour une raison simple : les capitales sont à peu près les mêmes en italique et en romain).

Curieusement, le package `smallcap` prévoit des petites capitales italiques (alors qu'elles n'existent pas dans la fonte), et ça produit ce résultat surprenant : *Petites Capitales Italiques*. En fait, il fait appel à une fonte qu'aucun autre package ne permet d'atteindre à ma connaissance : le télétype italique en largeur proportionnelle.

Pour comparer :
Télétype Italique en largeur fixe
Télétype Italique en largeur variable.

Il est amusant que ce package permette d'atteindre cette fonte très inhabituelle, et tout à fait regrettable qu'il permette de l'atteindre sous un faux nom.

2.18 Changement de corps relatif : le package `scalefnt`

C'est un des nombreux petits packages que nous devons à David CARLISLE. Il permet de changer le corps du texte[18], un peu comme les commandes `\small` ou `\large` que tu connais, mais de manière relative, *i.e.* « plus grand que le corps en cours, quel qu'il soit ».

En fait, ce tout petit package ne fournit qu'une unique commande, que tu pourras ensuite adapter à ta sauce, la commande `\scalefnt`, qui prend un seul argument : le facteur d'échelle. Une valeur de 1 indique de ne pas changer de taille, une valeur de 0,8 indique 80% de la taille en cours, et une valeur de 1,2 indique 120% de la taille en cours.

LATEX, comme c'est expliqué dans le passage sur NFSS (chapitre neuvième, au paragraphe 11 page 607), n'accepte de prendre que les tailles pour lesquelles les fontes sont définies. En standard, des tailles *discrètes* sont définies. Dans certains cas précis, des tailles *continues* sont définies. Dans le cas des tailles discrètes, si on est en corps 10, et qu'on demande à avoir 1,4 fois plus grand, on obtiendrait la taille définie la plus proche du corps 14. C'est en général du 14,4. C'est ce phénomène qui explique que dans l'exemple ci-après il arrive que deux lignes consécutives aient la même taille, alors que l'on s'attend à avoir une ligne 10% plus grosse que l'autre : certaines tailles intermédiaires ne sont pas définies.

Un petit exemple rapide :

```
\begin{center}
Texte!\\
\scalefont{1.1}Texte!\\
\scalefont{1.1}Texte!\\
\scalefont{1.1}Texte!\\
```

[18]La taille, si tu préfères.

```
\scalefont{1.1}Texte!\\
\scalefont{1.1}Texte!
\end{center}
```

produira :

Texte !
Texte !
Texte !
Texte !
Texte !
Texte !

Attention cependant, cette commande se comporte comme `\small`, c'est-à-dire que le changement de corps est valable jusqu'à nouvel ordre. Il n'existe pas d'équivalent en environnement.

Plus précisément, pour que ça puisse être utilisé comme un environnement, il faut faire appel à :

```
\let\endscalefont\empty
```

par exemple juste après avoir chargé le package. Pour plus d'infos sur ce que fait cette ligne un peu obscure, reporte-toi au chapitre neuvième page 571.

Et, du coup, on peut l'utiliser comme ça :

```
\begin{scalefont}{1.1}
Un peu gros.
\end{scalefont}
```

2.19 Forcer une taille : le package `textfit`

Le package `textfit`, que nous devons à Sebastian Rahtz et Phil Taylor [181], permet de choisir une taille telle que le texte prendra une place connue à l'avance, soit en hauteur, soit en largeur.

Le principe est simple : si on souhaite qu'un texte donné, par exemple un titre, vienne prendre une largeur choisie, le package se chargera de calculer quel corps de fonte il faut utiliser pour ce faire.

`\scaletowidth{`⟨*largeur*⟩`}{`⟨*texte*⟩`}`

`\scaletoheight{`⟨*hauteur*⟩`}{`⟨*texte*⟩`}`

Le package fournit deux commandes : la première, `\scaletowidth` calcule le corps de fonte nécessaire pour que le texte (deuxième paramètre) soit amené à la largeur demandée (premier paramètre) ; la seconde, `\scaletoheight` fait la même chose sur la hauteur du texte.

Ainsi, un appel à :

```
\scaletowidth{10cm}{Dix centimètres}
```

produira, par exemple, le résultat suivant[19] :

Dix centimètres

2.20 Interlettrage : le package `tracking`

Ce package, que nous devons à D. A. Glaskov, permet de faire de l'interlettrage[20]. L'interlettrage est une technique que je n'ai rencontrée que dans la presse quotidienne ; *elle est assez unanimement considérée comme une mauvaise pratique par les typographes*, sauf évidemment dans certains cas extrêmes dans lesquels il n'y a pas de bonne solution. Le principe consiste à mettre un peu de blanc entre les lettres des mots pour faire la justification. Le seul cas que je connaisse où ça puisse être vaguement toléré c'est quand, avec des colonnes trop étroites et un mot trop long on se retrouve avec ce mot tout seul sur sa ligne.

Par exemple, dans ce texte, la colonne ne fait que deux centimètres, ce qui n'est pas très raisonnable. Du coup, un mot se retrouve seul sur sa ligne si on laisse LATEX faire tout seul (colonne de gauche). En utilisant le package `tracking`, j'ai pu obtenir l'exemple de la colonne de droite[21] :

Ce texte est	Ce texte est
un peu long,	un peu long,
il fait plu-	il fait plu-
sieurs lignes,	sieurs lignes,
avec certains	avec certains
mots un peu	mots un peu
longs qui	longs qui
pourraient	p o u r r a i e n t
finalement se	finalement se
retrouver à	retrouver à
un par ligne.	un par ligne.

`\track{⟨largeur⟩}{⟨texte⟩}`

Cette commande ajoute la largeur indiquée en premier paramètre à chaque élément du texte indiqué en deuxième paramètre. Par exemple, pour ajouter un point à chaque lettre du mot lettre :

```
\track{1pt}{lettre}
```

qui produit ce résultat-là (très moche) : l e t t r e

[19] Une règle de 10 centimètres, graduée, a été placée sous le texte, pour pouvoir juger facilement du résultat.

[20] Le package `soul` (voir à ce sujet le paragraphe 2.45 page 150), propose la même fonctionnalité, et a l'avantage d'être mieux maintenu.

[21] Quelques remarques cependant sur le fait d'utiliser ce package. La première est qu'il définit une commande `\endlist`, pour son usage interne, et que cette commande entre en conflit avec la commande `\end{list}` de LATEX, ce qui est très ennuyeux (l'environnement `list` est très utilisé, l'air de rien, il se cache derrière presque tout ce qui ressemble à une liste). Ce conflit se lève en utilisant le package `contextsw` pour sauvegarder la définition de la commande `\endlist`.

Le deuxième point est que, dans certaines conditions, le package fait des erreurs de calcul : si tu as des ligatures (fi, ff, ffi) dans le texte où tu fais de l'interlettrage, certaines erreurs de calcul vont avoir lieu.

Une version du package corrigeant ces deux problèmes, `mytracking`, est disponible en ligne : `http://jmpl.fr.eu.org/JMPL/mytracking.sty`

`\fittrack{⟨largeur⟩}{⟨texte⟩}`

Cette commande ajoute la largeur nécessaire à chaque élément du texte passé en deuxième paramètre pour qu'il fasse la largeur indiquée en premier paramètre. C'est cette commande que j'ai utilisé dans l'exemple :

```
\fittrack{\linewidth}{pourraient}
```

`\ratiotrack{⟨ratio⟩}{⟨texte⟩}`

Cette commande ajoute la largeur nécessaire à chaque élément du texte passé en deuxième paramètre pour que sa longueur soit modifiée dans la proportion indiquée par le premier paramètre. Par exemple, pour qu'il soit 10% plus long :

```
\ratiotrack{1.1}{lettre}
```

qui produit ce résultat ci (toujours très moche) : lettre.

Je tiens à te le redire encore une fois : *il ne faut pas faire d'interlettrage*. Voilà. Tu es prévenu.

2.21 Changement de taille : le package `moresize`

Ce package, que nous devons à Christian Cornelssen [50], permet d'avoir des commandes supplémentaires pour changer de taille.

Ce package définit deux nouvelles tailles : `\ssmall`, qui est intermédiaire entre `\scriptsize` et `\tiny`, et `\HUGE` pour écrire vraiment très gros.

Le tableau 8.1 page 35 te présentait les tailles standard de LaTeX, le tableau 2.1, pour sa part, te présente le jeu de tailles enrichi par le package.

Commande	**Exemple**
`\tiny`	Exemple
`\ssmall`	Exemple
`\scriptsize`	Exemple
`\footnotesize`	Exemple
`\small`	Exemple
`\normalsize`	Exemple
`\large`	Exemple
`\Large`	Exemple
`\LARGE`	Exemple
`\huge`	Exemple
`\Huge`	Exemple
`\HUGE`	Exemple

Tab. 2.1: Changements de taille (`moresize`)

Ce package reconnaît plusieurs options. La première variété d'options qu'il reconnaît sont les options de taille (10pt, 11pt, et 12pt). Normalement, tu as passé l'une de ces options à `\documentclass` (par défaut, c'est 10pt), le package en hérite donc. C'est ce qui lui permettra de décider à quelle taille correspondent les nouvelles commandes, pour s'adapter à ton document.

La deuxième famille d'options porte sur la gestion des indices en mathématiques : si le package utilise la même règle que LaTeX pour choisir la taille des indices pour les très grandes tailles, ça fera des indices qui peuvent paraître trop grands. Par défaut, le package `moresize` respectera les habitudes de LaTeX. Avec l'option normalscripts, il réduira un peu la taille des indices pour la taille `\HUGE`. Si on lui passe l'option smallscripts, il fera des indices plus petits pour toutes les grandes tailles (au delà de `\LARGE`, en gros). Le rôle de normalscripts n'est pas clair, même après de nombreux essais, je n'ai pas réussi à produire un exemple parlant. Par contre l'option smallscripts produit un résultat qui est nettement plus intéressant que ce qu'on a par défaut.

2.22 Fonte de base plus grosse : le package `extsizes`

L'idée à l'origine de ce package, dont je ne connais pas l'auteur, est assez simple : les classes de base de LaTeX proposent comme taille de base pour les fontes, 10, 11 et 12 points. Tu pourrais avoir besoin d'autres tailles. Attention cependant : autant il peut être raisonnable de vouloir des fontes plus petites (pour un dictionnaire, par exemple), il semble *a priori* déraisonnable de vouloir utiliser des fontes plus grosses que le corps 12. En effet, 12 points, c'est déjà gros. L'usage de ce package pour obtenir une taille de base plus élevée est donc à éviter, sauf si tu en as *vraiment* besoin. Par exemple, si ton besoin c'est de pouvoir réduire le document à la photocopieuse après, ce n'est pas le bon package (voir le package `geometry`, au paragraphe 4.16 page 258, pour ça). Par contre, si tu as besoin d'écrire très gros parce que le document est destiné à des mal-voyants, alors ça peut être une bonne option.

Le package `extsizes` est accompagné de plusieurs classes de document que tu peux utiliser à la place des classes standard de LaTeX : la classe **extarticle** est équivalente à **article** ; **extreport** est équivalente à **report** ; **extbook** équivalente à **book**, **extproc** équivalente à **proc** ; et **extletter** équivalente à **letter**.

Toutes ces classes peuvent recevoir les options 8pt, 9pt, 10pt, 11pt, 12pt, 14pt, 17pt, et 20pt. Ces options permettent de fixer des tailles de base de texte plus importantes.

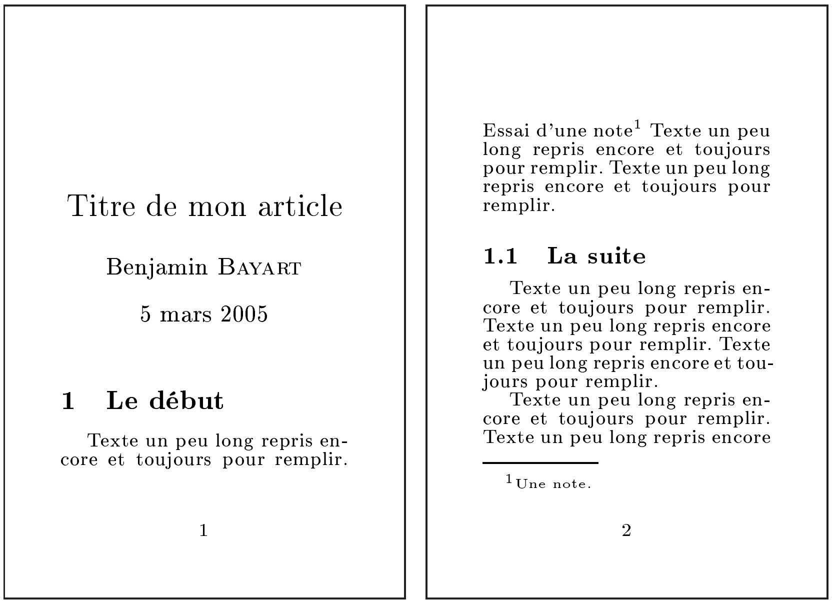
Si tu utilises les fontes Computer Modern, les classes se chargeront de faire les redéfinitions utiles pour pouvoir obtenir de très grandes fontes. Si tu utilises d'autres fontes, il se peut que les très grandes tailles ne soient pas disponibles. En règle générale, elles sont disponibles systématiquement pour les fontes PostScript, et souvent pour les autres fontes. Le plus simple est d'essayer.

Pour l'utilisation avec les classes standard de LaTeX, le simple changement de classe devrait suffire. Si tu voulais utiliser une autre classe, *a priori*, ça ne marche pas. Une solution, qui ne marche pas à tous les coups, est de charger le package `extsizes` en lui passant en option la taille de fonte que tu vises.

2.23 Document en 7 points : la classe viiptart

Cette classe, que nous devons à Moshe ZADKA [229], fonctionne exactement comme la classe **article** de LaTeX, sauf que la taille par défaut est de 7 points (eu lieu de 10 dans **article**), et que cette taille n'est pas modifiable. À utiliser si les classes fournies par les packages `extsizes` ne suffisent pas (ce package ne descend que jusqu'à 8 points).

```
* \documentclass{viiptart}
* \usepackage[T1]{fontenc}
* \usepackage[latin1]{inputenc}
* \makeatletter
* \def\@ptsize{0}
* \makeatother
* \usepackage[frenchb]{babel}
* % Utilisation de geometry
* % pour re-creer les marges
* % des exemples habituels.
* % \texte comme d'habitude.
  \title{Titre de mon article}
  \author{Benjamin
          \textsc{Bayart}}
  \date{\today}
  \maketitle
  \section{Le d\'ebut}
  \texte Essai d'une
  note\footnote{Une note.}
  \texte \texte \par
  % etc...
```



Titre de mon article

Benjamin Bayart

5 mars 2005

1 Le début

Texte un peu long repris encore et toujours pour remplir.

1

Essai d'une note[1] Texte un peu long repris encore et toujours pour remplir. Texte un peu long repris encore et toujours pour remplir.

1.1 La suite

Texte un peu long repris encore et toujours pour remplir. Texte un peu long repris encore et toujours pour remplir. Texte un peu long repris encore et toujours pour remplir.
Texte un peu long repris encore et toujours pour remplir. Texte un peu long repris encore

[1] Une note.

2

Il faut bien garder en tête que cet exemple est montré en grandeur nature : l'usage de `geometry` ne fait que modifier la taille du papier et des marges pour produire ce format de papier voisin du timbre poste, mais l'exemple n'est pas réduit pour être inclus ici.

À noter également les trois lignes magiques autour de la commande `\@ptsize` : cette commande est définie par les classes standard de LaTeX pour définir la taille par défaut. Elle vaut `0` pour indiquer du 10 points, `1` pour du 11 points et `2` pour du 12 points. Comme la classe **viiptart** ne passe pas par le même mécanisme d'options sur la taille, elle ne définit pas cette commande ; du coup les packages qui comptaient dessus ne fonctionnent plus. C'est entre autres le cas de l'option `frenchb` de `babel`.

La définition que je donne ici fait croire à `babel` que le document est en 10 points, c'est le mieux que je puisse faire : il n'y a pas de convention pour dire que le document est en 7 points.

Fig. 2.1: Exemple d'utilisation de la classe **viiptart**

L'équivalent pour les classes **book** ou **report** n'existe pas à ma connaissance. Ceci dit, cela ne relève pas d'une grande urgence : exactement comme ça semble une mauvaise idée d'aller au delà de 12 points, ça ne semble pas une bonne idée de descendre en dessous de 9 points, en dehors bien entendu d'applications très spécifiques (faire une anti-sèche, par exemple).

La figure 2.1 présente un exemple d'utilisation de ce package. Qu'on se comprenne bien : dans l'exemple, j'ai utilisé un format de papier (3 pouces de haut sur 2 pouces de large, assez voisin d'une carte de visite) tout petit, *je n'ai pas redimensionné le document pour l'inclure.* C'est-à-dire que la taille du texte est bien la taille que tu obtiendras avec la classe **viiptart** : tout petit. Il faut de bons yeux pour lire ça. À mon avis c'est à proscrire en dehors des dictionnaires et des anti-sèches.

2.24 Écriture scolaire : le package frcursive

Ce petit package, que nous devons à Emmanuel BEFFARA [13] permet de produire la forme de cursive à laquelle nous avons tous été habitués par nos instituteurs lorsque nous avons appris à écrire.

Il faut noter que cette fonte prétend être en encodage T1, ce qui n'est pas vrai : il manque énormément de caractères. Seules sont présentes les lettres accentuées utiles pour le français, et encore, pas toutes (il manque le Y-tréma majuscule, par exemple). D'autres caractères utiles manquent, comme les guillemets français. Cette limitation vient du fait que cette fonte fait appel à un très grand nombre de ligatures pour produire des caractères liés, et que ces ligatures prennent de la place dans les 256 symboles disponibles[22].

Cette fonte est fournie selon trois variantes.

La première variante est celle que l'on apprend à l'école. Ça donne ça, la lecture est simple. Très proche de ce qu'on avait l'habitude de lire et d'écrire à la petite école. *Ça existe en penché.* **Ça existe également en gras, *et en se donnant beaucoup de mal, on peut obtenir du gras-penché.*** Cela s'obtient en faisant appel à `\textcursive`, ou à `\cursive` (qui peut également être utilisé comme un environnement `cursive`). La variante penchée s'obtient avec `\textsl`, et la variante en gras avec `\textbf`, comme d'habitude[23].

Une souris verte, qui courrait dans l'herbe. Je l'attrape par la queue. Je la montre à ces Messieurs. Ces Messieurs me disent : trempez-la dans l'huile, trempez-la dans l'eau, vous aurez un escargot tout chaud.

La seconde variante est dite caligraphique. Elle accentue les pleins et les déliés, donnant un rendu plus soigné. Cela s'obtient en utilisant `\textcal` *dans* un passage en

[22]On pourrait envisager de fabriquer une fonte plus complète pour une extension de TEX admettant des fontes de plus de 256 symboles, mais ça n'a pas été fait.

[23]Il faut toutefois noter que la variante penché-gras (les deux à la fois) est censée exister, et est prévue par le package, mais la fonte n'est pas distribuée. On peut chercher à la produire, mais ce n'est pas très simple. Il vaut mieux décider de s'en passer. Pour obtenir le gras-penché, il faut appeler le script **mkdrv.sh** avec les paramètres **frcslbx** (le nom de base de la fonte), les trois modifieurs **bold**, **ext** et **slant**, pour les tailles 5, 7, 8, 9, 10, 10.95, 12, 14.4, 17.28, 20.74, 24.88 et 29.86. Ça donne cette ligne là :

```
sh ./mkdrv.sh frcslbx bold ext slant at 5 7 8 9 10 10.95 12 14.4 17.28 20.74 24.88 29.86
```

Ou encore, compléter le script en question en remplaçant les lignes suivantes (à la fin du script, tu trouveras facilement le bon endroit) :

```
make_series frcr                      at 5 7 8 9 10 10.95 12 14.4 17.28 20.74 24.88 29.86
make_series frcbx   bold ext          at 5 7 8 9 10 10.95 12 14.4 17.28 20.74 24.88 29.86
make_series frcsl   slant             at 5 7 8 9 10 10.95 12 14.4 17.28 20.74 24.88 29.86
make_series frcslbx bold ext slant    at 5 7 8 9 10 10.95 12 14.4 17.28 20.74 24.88 29.86
make_series frcc    callig            at 5 7 8 9 10 10.95 12 14.4 17.28 20.74 24.88 29.86
make_series frcf    fixed             at 5 7 8 9 10 10.95 12 14.4 17.28 20.74 24.88 29.86
```

puis appeler le script comme ça :

```
sh ./mkdrv.sh default
```

En effet, spontanément, le script ne génère que certaines tailles de certaines variantes de la fonte (les plus utilisées). Or, si tu essayes, avec le package, d'utiliser une taille et/ou une variante qui n'est pas produite par le script, et qui n'est donc pas sur ta machine, une erreur se produira. Il vaut donc mieux, par prudence, avoir toutes les variantes. On pourrait produire encore plus de variantes, par exemples la variante **fixed bold**, mais elle n'est pas prévue par le package, il faudrait donc également modifier le package.

cursive, ou en faisant appel à `\calseries`, ou à l'environnement `calseries`. Voyons quelques exemples[24] :

```
\textcursive{\textcal{Essai 1.}
{\cursive{\calseries Essai 2.}}
\begin{cursive}%
\begin{calseries}%
Essai 3.
\end{calseries}
\end{cursive}
```

produira : Essai 1. Essai 2. Essai 3.

Enfin, la troisième variante est celle dite « équiproportionnée », ou académique. Dans cette variante, la hauteur de chaque partie des lettres (montantes, descendantes, etc.) a été adaptée pour parfaitement s'aligner. Le rendu est assez différend (la fonte est plus étirée en hauteur et en profondeur), mais moins joli. En fait, il prend tout son sens dans ce cadre là :

Celle est la folie des hommes pressés.

On utilise `\textacad` *dans* un contexte cursive pour obtenir la variante académique, ou la commande `\acadshape`, ou encore l'environnement `acadshape`. La commande qui trace les traits est `\seyes`. Ces quelques exemples :

```
\textcursive{\textacad{Essai \seyes{d'alignement.}}}
\begin{cursive}
\begin{acadshape}
Alignement \seyes{essay\'e!}.
\end{acadshape}
\end{cursive}
```

produiront : Essai d'alignement. Alignement essayé!.

La commande `\seyes` peut être utilisée sur d'autres fontes, mais le résultat peut devenir étrange. Par exemple ici sur du Computer Modern. Ou encore sur de l'italique : *Essai en italique.*

Enfin, une dernière variante existe, moins jolie, qui est en largeur de trait fixe, c'est-à-dire que les pleins y sont aussi épais que les déliés. Elle n'a probablement d'intérêt que quand on veut modifier la fonte, pour pouvoir examiner les mouvements de crayon sans s'encombrer

[24] Le pourcent (%) à la fin du `\begin{cursive}` et du `\begin{calseries}` est important. Ces environnements ne provoquent pas de changement de ligne. S'ils sont appelés, comme ici, dans le cours d'une phrase, le changement de ligne après l'accolade fermante est traité comme un blanc (ce qui est normal pour un changement de ligne). TEX considère (avec raison) que n blancs consécutifs n'en font qu'un seul, mais pour cela il faut qu'ils soient *consécutifs*. Or dans l'exemple, les 3 blancs (un derrière chaque accolade fermante, correspondant au changement de ligne) ne sont *pas* consécutifs, ils sont séparés dans le source par des commandes, TEX ne va donc pas les aglomérer. Trois blancs, c'est deux de trop, il faut donc en supprimer deux. Pour les supprimer, il suffit de mettre en commentaire (avec un %) la fin de la ligne. D'où les deux % à conserver.

des fioritures. Le package la prévoit, puisqu'il donne les commandes nécessaires à LATEX pour la produire, mais ne la prévoit qu'à moitié, puisqu'il ne prévoit pas le commande pour l'utilisateur. Le résultat (peu intéressant) est celui-ci : En épaisseur fixe? Pour l'obtenir, il faut faire appel à :

```
\textcursive{\fontseries{ft}\selectfont En \'epaisseur fixe?}
```

2.25 Écriture scolaire : le package la

Ce package, que nous devons à Gerd Neugebauer [158], permet d'utiliser la fonte du même nom. C'est une fonte cursive, qui semble correspondre aux habitudes anglo-saxones en la matière. En particulier, il n'y a pas moyen d'avoir des caractères accentués[25].

Ce package définit en tout et pour tout six commandes et un environnement.

Les deux premières, `\textla`, et `\la`, permettent d'utiliser la fonte. Par exemple en appelant :

```
\textla{Text in a school-like handwriten font.}
```

on obtient : Text in a school-like handwriten font.

Les deux commandes suivantes, `\textlla` et `\lla`, utilisent une alternative à cette fonte qui contient, dessiné dans la fonte, les traits comme dans un cahier d'écolier. Ainsi, en appelant :

```
~~~\textlla{Text in a school-like handwriten font.}
```

on obtient : Text in a school-like handwriten font.

Enfin, les deux dernières commandes, `\llafill` et `\llaline` permettent de tracer les traits en question sans le texte. La première peut être utilisée pour remplir une ligne faite avec `\textlla` avec les traits. La seconde commande s'utilise comme `\underline`, mais au lieu de souligner trace des ligne de cahier d'ecolier. Un petit exemple :

```
\begin{center}
\llaline{A framed line\dots}

\textlla{Begin\llafill end.\linebreak
        Next line!}
\end{center}
```

qui produit :

A framed line...

Begin end.

Next line!

Enfin l'environnement `llapar` permet de taper un ou plusieurs paragraphes tels qu'on pourrait les voir dans un cahier d'écolier. Un exemple :

```
\begin{llapar}
Once upon a time, in a far far realm\dots
\end{llapar}
```

qui produit :

Once upon a time, in a far far realm...

[25] Pour ces deux raisons, à savoir l'absence des accents, et le dessin correspondant à une tradition anglaise, les exemples sont volontairement en anglais.

2.26 Écriture manuscrite : le package `twcal`

Ce package, que nous devons à Gene Neugebauer [160], permet d'utiliser la fonte TwCAL14 que nous devons à Thomas Weinmann. Il s'agit encore d'une fonte « manuscrite », mais cette fois-ci de tradition germanique. Les accents seront ..., sauf bien entendu le trêma.

Seules trois commandes sont définies. La première, et la moins intéressante, est `\twcallogo`, qui produit le logo TwCAL14.

Les deux autres, `\texttwcal` et `\twcal` permettent d'utiliser la fonte.

Un exemple rapide :

```
\begin{center}\twcal
Ein namhafter Wissenschaftler (man sagt, es sei Bertrand
Russell gewesen) hielt einen \"o ffentlichen Vortrag \"u
ber Astronomie. Er schilderte, wie die Erde um die Sonne
und die Sonne ihrerseits wieder um den Mittelpunkt einer
riesigen Ansammlung von Sternen kreist, die wir unsere
Galaxis nennen. Als der Vortrag beendet war, stand hinten
im Saal eine kleine alte Dame auf und erkl\"a rte:
\glqq Ich mu\ss{} Ihnen sagen, was Sie uns da erz\"a hlt
haben, stimmt alles nicht. In Wirklichkeit ist die Welt
eine flache Scheibe, die von einer Riesenschildkr\"o te
auf dem R\"u cken getragen wird.\grqq {} Mit einem \"u
berlegenen L\"a cheln hielt der Wissenschaftler ihr
entgegen: \glqq Und worauf steht die Schildkr\"o
te?\grqq{} -- \glqq Sehr schlau, junger Mann\grqq,
parierte die alte Dame. \glqq Ich werde es Ihnen sagen:
Da stehen lauter Schildkr\"o ten aufeinander.\grqq
\end{center}
```

produira :

Ein namhafter Wissenschaftler (man sagt, es sei Bertrand Russell gewesen) hielt einen öffentlichen Vortrag über Astronomie. Er schilderte, wie die Erde um die Sonne und die Sonne ihrerseits wieder um den Mittelpunkt einer riesigen Ansammlung von Sternen kreist, die wir unsere Galaxis nennen. Als der Vortrag beendet war, stand hinten im Saal eine kleine alte Dame auf und erklärte : „Ich muß Ihnen sagen, was Sie uns da erzählt haben, stimmt alles nicht. In Wirklichkeit ist die Welt eine flache Scheibe, die von einer Riesenschildkröte auf dem Rücken getragen wird." Mit einem überlegenen Lächeln hielt der Wissenschaftler ihr entgegen : „Und worauf steht die Schildkröte?" – „Sehr schlau, junger Mann", parierte die alte Dame. „Ich werde es Ihnen sagen : Da stehen lauter Schildkröten aufeinander."

2.27 Écriture manuscrite : le package `va`

Il existe une série de fontes `va`, et deux packages permettant de l'exploiter. Les deux packages s'appellent `va`. Et bien entendu, ils sont incompatibles l'un avec l'autre.

Le premier, livré avec la fonte, respecte la norme LaTeX, puisqu'il est accompagné d'un fichier de définition de fonte, mais le package en lui-même est dramatiquement mal écrit. De plus la définition de fonte utilise du 14 points à la place du 10, avec quelques risques qui en découlent. L'auteur de la fonte, comme de ce premier package est B. Ludewig.

Le second, fourni par Gerd Neugebauer, néglige de créer un fichier de définition de fonte, mais est par contre un petit peu mieux écrit (sans toutefois respecter la norme). C'est ce second package que je documente ici.

Cette fonte existe en deux variantes. La première est faite de traits d'une épaisseur fixe. On y accède avec les commandes `\va` ou `\textva`. La seconde, dite calligraphique, a une épaisseur de trait qui varie par endroit, ce qui donne un effet un peu plus élégant. On y accède avec les commandes `\vacal` et `\textvacal`.

Cette fonte correspond aux habitudes d'écriture manuscrite germanique, et donc ne contient pas d'autre accent que les quelques trêmas que l'auteur a bien voulu y inclure. Elle est cependant en codage `T1` pour permettre la césure des textes allemands.

Un petit exemple :

```
\textva{Exemple normal.}
\textvacal{Exemple calligraphique.}
```

qui produit : Exemple normal. Exemple calligraphique.

2.28 Écriture manuscrite : le package `pvscript`

Ce package, de Gerd Neugebauer [159], permet d'utiliser la fonte `script10.mf` de Peter Vanroose.

Cette fonte ne couvre pas l'ensemble des lettres, en particulier l'auteur n'y a pas inclus les accents, ce qui est curieux pour un belge.

Le package définit deux commandes : `\pvscript` et `\textpvscript`. On se contentera d'un petit exemple :

```
\textpvscript{Un petit exemple.}
```

qui donne : Un petit exemple.

2.29 Fonte calligraphique : le package `calligra`

Ce package, que nous devons à Gerd Neugebauer [157], permet d'utiliser une fonte qui n'est pas assez diffusée, mais pourtant jolie : la fonte `callig15`.

Cette fonte a deux défauts connus : l'espacement vertical de l'apostrophe (elle apparaît beaucoup trop haut), et l'espacement vertical du eszet (le ß) est trop large. Ce package corrige le deuxième problème, mais pas le premier.

Ce package fournit deux commandes de changement de fonte : `\textcalligra` et `\calligra`. Ça s'utilise par exemple ça :

```
\begin{center}\calligra
Cette fonte est parfois difficile \`a lire dans les petites tailles,
et le positionnement de l'apostrophe laisse \`a d\'esirer, c'est bien
triste. Stra\ss{e}.
\end{center}
```

et ça produit ce résultat-là[26] :

Cette fonte est parfois difficile à lire dans les petites tailles, et le positionnement de l'apostrophe laisse à désirer, c'est bien triste. Straße.

La commande `\textcalligra`, elle, s'utilise comme les commandes habituelles :

```
\textcalligra{essai}
```

et produit le même résultat : essai.

Pour résourdre le problème de l'apostrophe, j'ai écrit, il y a bien longtemps, le package `xavier`.

[26]Note l'apostrophe qui est placée tellement haut qu'on dirait un accent sur le « ℓ ».

	Avec `\app`	**Sans** `\app`
C'est ainsi	*C'est ainsi*	*C´est ainsi*
c'est ainsi	*c'est ainsi*	*c´est ainsi*
L'âge	*L'âge*	*L´âge*
l'âge	*l'âge*	*l´âge*

TAB. 2.2: Les problèmes d'apostrophe dans la fonte calligra

2.30 Fonte calligraphique : le package `xavier`

Ce package te permet de charger et d'utiliser une nouvelle famille de fontes (la famille `xav`) que Xavier[27] m'a demandé d'intaller pour son usage personnel.

Ce package est disponible à l'URL :
`http://jmpl.fr.eu.org/JMPL`
et se compose de deux fichiers : `xavier.sty` et `T1Liees.fd`.

Les seules commandes intéressantes sont :

```
\textxav{...}
{\xavfamily ...}
```

Cette fonte souffre de deux défauts : l'apostrophe est très mal placée. Après une lettre majuscule, elle a un rendu éventuellement acceptable, mais après une lettre minuscule, c'est catastrophique. Pour pallier ce problème, j'ai ajouté dans le package une petite commande `\app` qui remplace avantageusement l'apostrophe. Le tableau 2.2 te montrera quatre cas classiques d'usage de l'apostrophe.

Le deuxième défaut est le ß dont l'espacement est mauvais. J'ai repris dans le package `xavier` la solution proposée dans `calligra`.

Voyons maintenant un exemple d'utilisation de cette fonte :

```
\begin{center}\xavfamily
Telle est la folie des hommes press\'es de toujours
vouloir courir.

Cet exemple utilise l\app apostrophe comme il faut.
C\app est bizarre comme usage, mais c\app est ce que
j\app ai r\'eussi \`a faire de mieux.
Stra\ss{e}.
\end{center}
```

produit ce résultat :

Telle est la folie des hommes pressés de toujours vouloir courir.

Cet exemple utilise l'apostrophe comme il faut. C'est bizarre comme usage, mais c'est ce que j'ai réussi à faire de mieux. Straße.

[27] Un ami un peu fou qui voulait utiliser cette fonte bizarre pour taper certains textes.

2.31 Symboles en mode texte : le package `textcomp`

Les fontes « EC », que l'on utilise souvent de nos jours dans les documents LATEX, sont normalement livrées avec des fontes dites « TC », pour « Text Companion ». Ces fontes définissent un certain nombre de symboles utiles, qui ne sont pas dans les fontes classiques.

Le package `textcomp`, livré en standard avec LATEX, fournit les symboles des tableaux 2.3 à 2.9 pages 123–125.

Le symbole euro, présenté parmi les monnaies (tableau 2.3) est particulièrement laid. On en présentera d'autres variantes plus agréables, en particulier le symbole fourni par le package `china2e`, présenté au paragraphe 2.34 page 128.

Commande	Sym.
`\texteuro`	€
`\textdollar`	$
`\textcent`	¢
`\textyen`	¥
`\textpeso`	₱
`\textwon`	₩
`\textnaira`	₦
`\textbaht`	฿
`\textcurrency`	¤

Commande	Sym.
`\textsterling`	£
`\textdollaroldstyle`	$
`\textcentoldstyle`	¢
`\textlira`	₤
`\textflorin`	ƒ
`\textdong`	₫
`\textguarani`	₲
`\textcolonmonetary`	₡

TAB. 2.3: Symboles monétaires (package `textcomp`)

Commande	Sym.
`\textsection`	§
`\textreferencemark`	※
`\textdagger`	†
`\textbullet`	•
`\textborn`	⋆
`\textmarried`	⚭
`\textblank`	␢
`\textasteriskcentered`	∗
`\textsurd`	√

Commande	Sym.
`\textparagraph`	¶
`\textpilcrow`	¶
`\textdaggerdbl`	‡
`\textopenbullet`	◦
`\textdied`	†
`\textdivorced`	⚮
`\textmusicalnote`	♪
`\textleaf`	🍃

TAB. 2.4: Symboles typographiques (package `textcomp`)

Commande	**Sym.**	**Commande**	**Sym.**
\textquotestraightbase	‚	\textquotestraightdblbase	„
\texttwelveudash	—	\textthreequartersemdash	—
\textdblhyphen	⹀	\textdblhyphenchar	⹀
\textasciiacute	´	\textasciigrave	ˋ
\textgravedbl	˵	\textacutedbl	˝
\textasciibreve	˘	\textasciimacron	¯
\textasciidieresis	¨	\textasciicaron	ˇ
\textquotesingle	'	\texttildelow	˷
\textinterrobang	‽	\textinterrobangdown	⸘
\textlquill	⁅	\textrquill	⁆
\textbardbl	‖	\textbrokenbar	¦
\textperiodcentered	·		

TAB. 2.5: Ponctuation et accents (package `textcomp`)

Commande	**Sym.**	**Commande**	**Sym.**
\textzerooldstyle	0	\textoneoldstyle	1
\texttwooldstyle	2	\textthreeoldstyle	3
\textfouroldstyle	4	\textfiveoldstyle	5
\textsixoldstyle	6	\textsevenoldstyle	7
\texteightoldstyle	8	\textnineoldstyle	9
\textonesuperior	^{1}	\texttwosuperior	^{2}
\textthreesuperior	^{3}	\textonequarter	$\frac{1}{4}$
\textonehalf	$\frac{1}{2}$	\textthreequarters	$\frac{3}{4}$
\textordfeminine	ª	\textordmasculine	º

TAB. 2.6: Symboles pour des quantités (package `textcomp`)

Commande	**Sym.**	**Commande**	**Sym.**
\textfractionsolidus	⁄	\textminus	−
\textlangle	⟨	\textrangle	⟩
\textperthousand	‰	\textpertenthousand	‱
\textnumero	№	\textbigcircle	◯
\textlbrackdbl	⟦	\textrbrackdbl	⟧
\textuparrow	↑	\textdownarrow	↓
\textleftarrow	←	\textrightarrow	→
\textpm	±	\textlnot	¬
\texttimes	×	\textdiv	÷

TAB. 2.7: Symboles pseudo-mathématiques (package `textcomp`)

Commande	Sym.
`\textmho`	℧
`\textcelsius`	℃
`\textmu`	µ

Commande	Sym.
`\textohm`	Ω
`\textdegree`	°

TAB. 2.8: Symboles d'unités (package `textcomp`)

Commande	Sym.
`\textcopyleft`	🄯
`\textcircledP`	Ⓟ
`\texttrademark`	™
`\textrecipe`	℞
`\textestimated`	℮

Commande	Sym.
`\textcopyright`	©
`\textregistered`	®
`\textservicemark`	℠
`\textdiscount`	⁒

TAB. 2.9: Symboles et logotypes divers (package `textcomp`)

2.32 Les « Zapf dingbats » : le package `bbding`

Ce petit package, qui ne contient que des symboles et ne sert qu'à gérer la fonte `bbding`, nous est offert par Karel HORAK et Peter Møller NEERGAARD [155].

Les symboles de ce package sont présentés dans les tableaux 2.10 à 2.17 pages 125–127.

Commande	Sym.
`\ScissorLeft`	✂
`\ScissorLeftBrokenTop`	✁
`\ScissorLeftBrokenBottom`	✃
`\ScissorHollowLeft`	✄

Commande	Sym.
`\ScissorRight`	✂
`\ScissorRightBrokenTop`	✁
`\ScissorRightBrokenBottom`	✃
`\ScissorHollowRight`	✄

TAB. 2.10: Ciseaux (package `bbding`)

Commande	Sym.
`\HandLeft`	☜
`\HandCuffLeft`	☚
`\HandLeftUp`	☝
`\HandCuffLeftUp`	☝
`\HandPencilLeft`	✍

Commande	Sym.
`\HandRight`	☞
`\HandCuffRight`	☛
`\HandRightUp`	☝
`\HandCuffRightUp`	☝

TAB. 2.11: Mains (package `bbding`)

Commande	Sym.
\PencilLeft	
\PencilLeftUp	
\PencilLeftDown	
\NibLeft	
\NibSolidLeft	

Commande	Sym.
\PencilRight	✏
\PencilRightUp	✐
\PencilRightDown	✎
\NibRight	✑
\NibSolidRight	✒

TAB. 2.12: Stylos (package bbding)

Commande	Sym.
\XSolid	✕
\Plus	✚
\PlusCenterOpen	✜
\Cross	✝
\CrossOutline	✞
\CrossMaltese	✠
\XSolidBrush	✗

Commande	Sym.
\XSolidBold	✖
\PlusOutline	✙
\PlusThinCenterOpen	✛
\CrossBoldOutline	✞
\CrossOpenShadow	✟
\CrossClowerTips	✣

TAB. 2.13: Croix diverses (package bbding)

Commande	Sym.
\DavidStar	✡
\JackStar	✢
\FourStar	✦
\FiveStar	★
\FiveStarOpen	☆
\FiveStarCenterOpen	✫
\FiveStarOutline	✬
\FiveStarConvex	✭
\SixStar	✶
\EightStar	✷
\EightStarTaper	✴
\TwelweStar	✹
\AsteriskThin	✳
\AsteriskBold	✱
\Asterisk	✽
\FourAsterisk	✢

Commande	Sym.
\DavidStarSolid	✶
\JackStarBold	✤
\FourStarOpen	✧
\FiveStarLines	✰
\FiveStarOpenCircled	✪
\FiveStarOpenDotted	✮
\FiveStarOutlineHeavy	✯
\FiveStarShadow	✰
\EightStarBold	✸
\EightStarConvex	✵
\SixteenStarLight	✺
\AsteriskThinCenterOpen	✲
\AsteriskCenterOpen	✼
\AsteriskRoundedEnds	❃
\EightAsterisk	❊

TAB. 2.14: Étoiles variées (package bbding)

Commande	Sym.	Commande	Sym.
\FiveFlowerOpen	❀	\FiveFlowerPetal	✿
\SixFlowerOpenCenter	✲	\SixFlowerRemovedOpenPetal	❁
\SixFlowerAlternate	✾	\SixFlowerAltPetal	❃
\SixFlowerPetalDotted	❁	\SixFlowerPetalRemoved	✼
\EightFlowerPetal	❂	\EightFlowerPetalRemoved	❊
\FourClowerOpen	☘	\FourClowerSolid	☘
\Sparkle	❇	\SparkleBold	❈
\Snowflake	❄		
\SnowflakeChevron	❆	\SnowflakeChevronBold	❆

TAB. 2.15: Fleurs et flocons (package bbding)

Commande	Sym.	Commande	Sym.
\CircleSolid	●	\CircleShadow	❍
\HalfCircleLeft	◖	\HalfCircleRight	◗
\Ellipse	⬭	\EllipseSolid	⬬
\EllipseShadow	⬭		
\Square	□	\SquareSolid	■
\SquareShadowTopLeft	❐	\SquareShadowTopRight	❐
\SquareShadowBottomRight	❏		
\SquareCastShadowTopLeft	❐	\SquareCastShadowTopRight	❒
\TriangleUp	▲	\TriangleDown	▼
\DiamondSolid	◆	\OrnamentDiamondSolid	❖
\Rectangle	❙		
\RectangleThin	❘	\RectangleBold	❚

TAB. 2.16: Formes géométriques (package bbding)

Commande	Sym.	Commande	Sym.
\Phone	☎	\PhoneHandset	✆
\Tape	✇	\Peace	✌
\Plane	✈	\Envelope	✉
\Checkmark	✓	\CheckmarkBold	✔
\ArrowBoldRightStrobe	➠	\ArrowBoldRightCircled	➲
\ArrowBoldUpRight	➦	\ArrowBoldDownRight	➥
\ArrowBoldRightShort	➧	\SunshineOpenCircled	❂

TAB. 2.17: Symboles divers (package bbding)

Bois		Feu		Terre		Fer		Eau		
1		13		25		37		49		子 Rat
	2		14		26		38		50	丑 Bœuf
51		3		15		27		39		寅 Tigre
	52		4		16		28		40	卯 Lapin
41		53		5		17		29		辰 Dragon
	42		54		6		18		30	巳 Serpent
31		43		55		7		19		午 Cheval
	32		44		56		8		20	未 Mouton
21		33		45		57		9		申 Singe
	22		34		46		58		10	酉 Coq
11		23		35		47		59		戌 Chien
	12		24		36		48		60	亥 Cochon
甲	乙	丙	丁	戊	己	庚	辛	壬	癸	
1	2	3	4	5	6	7	8	9	10	

TAB. 2.18: Le cycle de 60 ans du calendrier chinois (花甲子)

2.33 Rapport d'aspect : le package `ar`

Ce tout petit package de rien du tout est écrit par Claudio BECCARI [12]. Il fournit un unique symbole : AR, que l'on obtient en appelant la commande `\AR`. Il semble que dans certaines contrées on utilise ce symbole pour indiquer le rapport d'aspect (« Aspect Ratio », en anglais).

En maths cependant :

$$\mathrm{AR} = \mathrm{AR}$$

Ce package peut poser problème dans un document utilisant beaucoup de maths : il consomme un des 16 alphabets mathématiques disponibles dans TeX, pour un seul symbole. Pour contourner ce problème, se reporter au chapitre dixième page 629.

2.34 Chinois pour rire : le package `china2e`

Ce package, que nous devons à Udo HEYL [86], n'est pas une implémentation sérieuse du chinois, mais permet juste l'utilisation d'une fonte qui contient quelques caractères « pour rire ». C'est, par exemple, suffisant pour obtenir les symboles de l'astrologie chinoise, mais clairement pas suffisant pour un usage sérieux.

Sinophones et autres sinophiles, s'abstenir, donc.

2.34.1 Le calendrier chinois

Le tableau 2.18 présente les 60 ans du cycle du calendrier chinois. Le premier cycle a débuté en 2637 avant notre ère. Le tableau présente le nom de chacune des années du cycle. Ainsi, la 14$^{\text{ième}}$ année d'un cycle s'écrit 丁丑.

2004 est l'année 21 du 78$^{\text{ième}}$ cycle, et s'écrit donc 甲申 (Singe de bois).

Commande	Résultat	Commande	Résultat
\TerrEle{1}	子	\terrele{1}	子
\TerrEle{2}	丑	\terrele{2}	丑
\TerrEle{3}	寅	\terrele{3}	寅
\TerrEle{4}	卯	\terrele{4}	卯
\TerrEle{5}	辰	\terrele{5}	辰
\TerrEle{6}	巳	\terrele{6}	巳
\TerrEle{7}	午	\terrele{7}	午
\TerrEle{8}	未	\terrele{8}	未
\TerrEle{9}	申	\terrele{9}	申
\TerrEle{10}	酉	\terrele{10}	酉
\TerrEle{11}	戌	\terrele{11}	戌
\TerrEle{12}	亥	\terrele{12}	亥

TAB. 2.19: Les 12 éléments terrestres du calendrier chinois

Commande	Résultat	Commande	Résultat
\AstrEle{1}	甲	\astrele{1}	甲
\AstrEle{2}	乙	\astrele{2}	乙
\AstrEle{3}	丙	\astrele{3}	丙
\AstrEle{4}	丁	\astrele{4}	丁
\AstrEle{5}	戊	\astrele{5}	戊
\AstrEle{6}	己	\astrele{6}	己
\AstrEle{7}	庚	\astrele{7}	庚
\AstrEle{8}	辛	\astrele{8}	辛
\AstrEle{9}	壬	\astrele{9}	壬
\AstrEle{10}	癸	\astrele{10}	癸

TAB. 2.20: Les 10 éléments astraux du calendrier chinois

2.34.2 Le calendrier : symboles

Sinon, si tu veux quelques symboles chinois, tu peux te reporter aux tableaux suivants :
- les éléments terrestres, tableau 2.19 ;
- les éléments astraux, tableau 2.20 ;
- les positions de la lune, tableau 2.21 page suivante ;
- les phases de la lune, tableau 2.22 page suivante.

Le cycle de 60 ans du calendrier chinois s'obtient avec `\CyclYears` (花甲子), l'année s'écrit `\Year` (年), le mois s'écrit `\Month` (月) et le jour `\Day` (日). Le calendrier chinois (livre de mille ans) est désigné par `\Thousand\Year\Book` (千年书). Les 28 positions de la lune sont désignées par `\MoonStations` (宿), les 15 premiers jours d'un signe du zodiaque (le montant) par `\WaxingZodiac`, les 15 derniers (le descendant) par `\WaningZodiac`, un signe du zodiaque se dit `\ZodiacSign` (節氣).

Les 12 éléments terrestres sont désignés par `\TerrElements` (地支) et les 10 astraux par `\AstrElements` (天干). Le soleil (positif, mâle) est désigné par `\Solar` (阳), la lune (négatif, femelle) est désignée par `\Lunar` (阴).

Commande	Résultat	Commande	Résultat
\MoonSta{1}	危	\moonsta{1}	危
\MoonSta{2}	室	\moonsta{2}	室
\MoonSta{3}	壁	\moonsta{3}	壁
\MoonSta{4}	奎	\moonsta{4}	奎
\MoonSta{5}	婁	\moonsta{5}	婁
\MoonSta{6}	胃	\moonsta{6}	胃
\MoonSta{7}	昴	\moonsta{7}	昴
\MoonSta{8}	畢	\moonsta{8}	畢
\MoonSta{9}	嘴	\moonsta{9}	嘴
\MoonSta{10}	参	\moonsta{10}	参
\MoonSta{11}	井	\moonsta{11}	井
\MoonSta{12}	鬼	\moonsta{12}	鬼
\MoonSta{13}	柳	\moonsta{13}	柳
\MoonSta{14}	星	\moonsta{14}	星
\MoonSta{15}	張	\moonsta{15}	張
\MoonSta{16}	翼	\moonsta{16}	翼
\MoonSta{17}	軫	\moonsta{17}	軫
\MoonSta{18}	角	\moonsta{18}	角
\MoonSta{19}	亢	\moonsta{19}	亢
\MoonSta{20}	氐	\moonsta{20}	氐
\MoonSta{21}	房	\moonsta{21}	房
\MoonSta{22}	心	\moonsta{22}	心
\MoonSta{23}	尾	\moonsta{23}	尾
\MoonSta{24}	箕	\moonsta{24}	箕
\MoonSta{25}	斗	\moonsta{25}	斗
\MoonSta{26}	牛	\moonsta{26}	牛
\MoonSta{27}	女	\moonsta{27}	女
\MoonSta{28}	虚	\moonsta{28}	虚

TAB. 2.21: Les 28 positions de la lune du calendrier chinois

Commande	Résultat
\MoonPha{1}	🌚
\MoonPha{2}	🌛
\MoonPha{3}	🌝
\MoonPha{4}	🌜

TAB. 2.22: Les 4 phases de la lune du calendrier chinois

Commande	Rés.	Commande	Rés.	Commande	Rés.
`\Nul` (0)	零	`\Ten` (10)	十	`\Twenty` (20)	二十
`\One` (1)	一	`\Eleven` (11)	十一	`\Thirty` (30)	三十
`\Two` (2)	二	`\Twelve` (12)	十二	`\Forty` (40)	四十
`\Three` (3)	三	`\Thirteen` (13)	十三	`\Fifty` (50)	五十
`\Four` (4)	四	`\Fourteen` (14)	十四	`\Sixty` (60)	六十
`\Five` (5)	五	`\Fifteen` (15)	十五	`\Seventy` (70)	七十
`\Six` (6)	六	`\Sixteen` (16)	十六	`\Eighty` (80)	八十
`\Seven` (7)	七	`\Seventeen` (17)	十七	`\Ninety` (90)	九十
`\Eight` (8)	八	`\Eighteen` (18)	十八	`\Hundred` (100)	百
`\Nine` (9)	九	`\Nineteen` (19)	十九	`\Thousand` (1000)	千

TAB. 2.23: Numération chinoise selon `china2e`

Commande	Résultat
`\vdots`	⋮
`\Euro`	€
`\Greenpoint`	
`\Info`	$\boxed{i}$
`\Request`	$\boxed{?}$
`\Postbox`	✉
`\Pound`	
`\Telephone`	☎

TAB. 2.24: Symboles supplémentaires du package `china2e`

Le nouveau mois est désigné par `\New\Month` (新月) et le bisextile par `\Leap` (闰).

Le nouvel an grégorien est désigné par `\NewGregYear` (元旦) et le nouvel an chinois par `\NewChinYear` (新年). Le calendrier lunaire chinois est représenté par `\Lunar\Calendar` (阴历).

Le bois, `\Wood` (木), correspond aux 2 premiers éléments astraux, le feu, `\Fire` (火), aux 2 suivants, la terre, `\Earth` (土), aux 2 suivants, le métal, `\Metal` (金), aux 2 suivants et enfin l'eau, `\Water` (水), correspond aux 2 derniers éléments astraux.

2.34.3 La numération

Les commandes utiles pour la numération en chinois font l'objet du tableau 2.23.

Le premier mois lunaire se désigne par `\FirstMonth` (正月), le premier mois du calendrier grégorien (janvier, hein, faut pas se laisser déboussoler), se désigne sottement par `\One\Month` (一月). Pour les autres mois, ils sont désignés de la même façon quel que soit le calendrier, par exemple pour le deuxième mois, `\Two\Month` (二月).

2.34.4 Symboles supplémentaires

En fait, de mon point de vue (la seule chose intéressante dans l'astrologie, ce sont les petits symboles rigolos), la commande la plus utile de ce package est la commande `\Euro`,

Commande	Résultat	Commande	Résultat
\symA	$\mathbb{A}$	\symN	$\mathbb{N}$
\symB	$\mathbb{B}$	\symO	$\mathbb{O}$
\symC	$\mathbb{C}$	\symP	$\mathbb{P}$
\symD	$\mathbb{D}$	\symQ	$\mathbb{Q}$
\symE	$\mathbb{E}$	\symR	$\mathbb{R}$
\symF	$\mathbb{F}$	\symS	$\mathbb{S}$
\symG	$\mathbb{G}$	\symT	$\mathbb{T}$
\symH	$\mathbb{H}$	\symU	$\mathbb{U}$
\symI	$\mathbb{I}$	\symV	$\mathbb{V}$
\symJ	$\mathbb{J}$	\symW	$\mathbb{W}$
\symK	$\mathbb{K}$	\symX	$\mathbb{X}$
\symL	$\mathbb{L}$	\symY	$\mathbb{Y}$
\symM	$\mathbb{M}$	\symZ	$\mathbb{Z}$
\Natural*	$\mathbb{N}$	\NATURAL*	N
\Integer*	$\mathbb{Z}$	\INTEGER*	Z
\Rational*	$\mathbb{Q}$	\RATIONAL*	Q
\Real*	$\mathbb{R}$	\REAL*	R
\Complex*	$\mathbb{C}$	\COMPLEX*	C
\symAE	$\ddot{\mathbb{A}}$	\symUE	$\ddot{\mathbb{U}}$
\symOE	$\ddot{\mathbb{O}}$		

* En mode maths uniquement.

TAB. 2.25: « Alphabet » de symboles maths proposé par `china2e`

---

qui fournit un symbole € tout à fait convenable. En tous cas beaucoup plus intéressant que celui présenté par le package `textcomp` (voir à ce sujet le paragraphe 2.31 page 123).

Les autres symboles supplémentaires sont présentés dans le tableau 2.24 page précédente.

2.34.5 Encore des maths

Le package `china2e` fournit une version de plus des lettres à double barre. Ces lettres s'obtiennent avec les commandes `\symA` ($\mathbb{A}$) à `\symZ` ($\mathbb{Z}$). Le tableau 2.25 présente l'ensemble de ces symboles, qui sont utilisables en mode texte.

Les symboles `\Natural`, `\Complex`, etc., sont, eux, utilisables en mode maths uniquement.

2.34.6 Pseudo manuscrit

Enfin, cette fonte contient des caractères pseudo-manuscrits en majuscules, tout justes utilisables pour l'anglais et l'allemand (il n'y a pas de lettres accentuées, à part les trèmas pour l'allemand).

Ça s'utilise avec la commande `\BLOCK` comme une fonte. Par exemple, en écrivant :

```
\begin{center}
{\BLOCK ESSAI AMUSANT. \AE~\UE~\OE.}
\end{center}
```

Times :

Les députés qui vont examiner, le 14 avril, en seconde lecture, la refonte de la loi Informatique et libertés ont l'intention de sévir. Dans sa première mouture, leur texte prévoit des peines de 5 ans d'emprisonnement et 300 000 € d'amende (contre 3 ans et 45 000 € jusqu'à présent) pour toute mise en œuvre d'un fichier sans déclaration à la Commission nationale informatique et liberté (Cnil). Et ce même par simple « négligence ».
Bricolage : €, €, €.

Concrete :

Les députés qui vont examiner, le 14 avril, en seconde lecture, la refonte de la loi Informatique et libertés ont l'intention de sévir. Dans sa première mouture, leur texte prévoit des peines de 5 ans d'emprisonnement et 300 000 € d'amende (contre 3 ans et 45 000 € jusqu'à présent) pour toute mise en œuvre d'un fichier sans déclaration à la Commission nationale informatique et liberté (Cnil). Et ce même par simple « négligence ».
Bricolage : €, €, €.

Computer Moderne :

Les députés qui vont examiner, le 14 avril, en seconde lecture, la refonte de la loi Informatique et libertés ont l'intention de sévir. Dans sa première mouture, leur texte prévoit des peines de 5 ans d'emprisonnement et 300 000 € d'amende (contre 3 ans et 45 000 € jusqu'à présent) pour toute mise en œuvre d'un fichier sans déclaration à la Commission nationale informatique et liberté (Cnil). Et ce même par simple « négligence ».
Bricolage : €, €, €.

Ce texte, d'un intérêt douteux, est emprunté, sans autorisation, au numéro 4355, daté du mercredi 14 avril 2004 du journal « Le Canard enchaîné ». Le texte original comportait le mot « euro » là où j'y ai introduit le symbole produit par le package `eurosym`.

FIG. 2.2: Trois textes utilisant le symbole euro du package `eurosym`

---

on obtient :

ESSAI AMUSANT. Ä Ü Ö.

À noter cependant : les commandes `\OE` et `\AE` écrasent les définitions standard de ces commandes et interdisent donc l'usage de Œ et Æ. Reporte-toi au chapitre dixième, paragraphe 4.3 page 638 pour plus de détails.

2.35 Produire le symbole €

Le problème peut te sembler trivial, mais il y a un nombre invraissemblable de solutions différentes, et pas toujours élégantes.

La solution la plus élégante, c'est bien évidemment d'utiliser le symbole prévu en standard par LaTeX, dans le package `textcomp` (solution déjà décrite au paragraphe 2.31 page 123). Ce package fournit la commande `\texteuro`, qui produit le symbole €. Ce symbole est bien laid, à mon grand regret. En sans-serif, il est presque acceptable : **€**.

Une solution moins élégante, mais au résultat nettement plus heureux, est de faire appel au package `china2e` (voir le paragraphe 2.34 page 128) qui n'a rien à voir avec le sujet, mais propose une commande `\Euro` qui produit le symbole €.

2.35.1 Le package `eurosym`

Nous devons ce package (et les fontes qui vont avec) à Henrik THEILING [204]. Il produit le symbole tel qu'il est définit par les textes officiels sur le sujet.

L'immense avantage par rapport à `textcomp`, c'est que le symbole est presque joli, et que par rapport à `china2e`, il est utilisable en gras, en italique, etc. Il reste cependant que le symbole produit ne cadre pas tout à fait avec le niveau de gris moyen d'un texte en « Computer Modern » (la fonte par défaut de LaTeX).

Ce package fournit deux commandes clefs, et quelques commandes annexes. Les deux commandes clefs sont `\euro`, qui produit le symbole €, et `\EUR`, qui prend un argument et produit 12,5 €. Le package admet une option left qui lui indique que le symbole monétaire doit être à gauche du montant, et une option right qui lui indique le contraire. Par défaut, l'option left est choisie, parce qu'elle cadre bien avec la langue anglaise, qui est celle par défaut de LaTeX. Amusant quand on pense qu'il n'y a *pas* de pays anglophone qui utilise l'euro.

Le symbole produit ressemble à une hérésie typographique : étant indépendant de la fonte en cours d'utilisation, il sera, selon le contexte, soit trop noir, comme c'est le cas en Computer Modern, soit convenable, comme c'est le cas pour Concrete, soit trop clair, comme c'est le cas en Times. La figure 2.2 page précédente illustre ce phénomène. De plus, ce symbole est en « sans serif », ce qui fait bizarre dans un texte normal.

Si vraiment on le trouve inadapté, le package permet de bricoler un symbole euro à partir du C majuscule de la fonte en cours d'utilisation (c'est en effet une méhode préconisée par les textes européens qui se mèlent décidément de ce qui ne les regarde pas). Le symbole bricolé peut être utilisé en trois variantes : avec une double barre courte, moyenne, ou longue. Les trois commandes correspondantes sont `\geneuronarrow` (€), `\geneuro` (€), et `\geneurowide` (€). Les versions bricolées sont également visibles dans le texte de la figure 2.2.

2.35.2 Le package `eurosans`

Ce minuscule package, que nous devons à Walter Schmidt, permet d'utiliser les fontes fournies par Adobe pour produire le symbole euro. Ces fontes sont gratuites, mais soumises à une license qui ne permet pas de les diffuser librement. C'est pourquoi je te déconseille *fermement* de t'en servir : en utilisant ces fontes, tu imposeras à ceux à qui tu diffuseras ton document de se plier aux conditions de la license Adobe. Bien que ces conditions ne soient pas très strictes, elles sont pénibles : puisqu'il n'est pas permis de diffuser telles quelles les fontes, leur installation ne peut pas être convenablement automatisée, et ces fontes ne sont pas prévues pour TeX.

Ce package fournit une unique commande, `\euro`, qui produit le symbole €.

Les deux symboles fournis respectivement par `eurosym` et `eurosans` peuvent sembler identique. Il n'en est rien, ils sont juste très similaires. Les voilà tous les deux, en suffisament grande taille, pour pouvoir regarder les détails. La différence ? Si on y regarde très attentivement, la variante `eurosans` est très légèrement moins haute, et très légèrement plus sombre (ça peut ne pas être évident sur un document sorti d'une imprimante de qualité moyenne).

`eurosym`	`eurosans`
€	€

2.35.3 Le package `europs`

Ce package, que nous devons à Reinhard Zierke, fait lui aussi usage des fontes fournies par Adobe, aussi je te déconseille également de l'utiliser.

Son principal intérêt, par rapport à `eurosans`, est qu'il utilise la variété « avec serif », bien qu'elle ne soit pas conforme du tout aux directives européennes en la matière.

Je trouve le résultat un peu moins satisfaisant que le symbole de `china2e`, mais plus homogène avec un texte classique que les variétés sans-serif.

La seule commande utile est `\EUR`, qui produit le symbole €. Le symbole s'adaptera au contexte, ainsi en sans-serif, il produit €, *en italique, il produit €*, **en gras, il produit €**, etc.

Ces fontes étant produites par Adobe, elles sont plus adaptées à un environnement en Times qu'à un environnement en Computer Modern.

2.35.4 Le package `eurofont`

Le package `eurofont` est plutôt à proscrire, c'est pourquoi je ne le documenterai pas. Ce package, pas si petit que ça, est l'œuvre de Rowland McDonnel [132].

Ce package cherche à combiner, selon la méthode qu'il juge la plus intelligente (et qui est configurable) avec tous les packages installés sur ton système pour savoir *quelle* variété de symbole euro il va produire.

Cela me pose deux problèmes. Le premier est que ça fait des documents qui ne sont pas portables : si tu écrit ton document en décidant d'utiliser le € fournit par `china2e`, que tu l'envoies à quelqu'un, il est possible, et même probable, que chez ce quelqu'un il ressorte avec le € fournit par `textcomp`. C'est un comportement que les habitués des logiciels soit-disant wysiwyg trouvent peut-être convenable, mais pour ma part je n'apprécie guère.

Le second problème est que l'aspect profondément versatile de ce package fait que si je te donne des exemples, je ne pourrai pas être certain que ces exemples donneront le même résultat pour toi, ce qui me compliquerait considérablement la tâche, et te compliquerait également la vie : chaque exemple serait en pareil cas assorti d'un ou plusieurs fichiers de configuration indiquant exactement quelles manipulations j'ai dû faire pour parvenir au résultat présenté.

Enfin, ce n'est pas un problème mais c'est tout de même gènant : ce package est bien compliqué, pour produire un seul symbole : 60 pages de mode d'emploi, ça en impose.

2.36 Logo METAFONT : le package `mflogo`

Ce package, que nous devons à Ulrick Vieth [214] nous permet d'utiliser les fontes qui correspondent au logo de METAFONT. En effet, le logo METAFONT est traditionnellement écrit dans cette fonte qui comporte d'ailleurs juste assez de caractères pour faire ce logo et celui d'un programme associé METAPOST.

Ce package crée quatre commandes utiles : `\MP` et `\MF` pour produire les deux logos que tu viens de voir, et `\textlogo` et `\logofamily` pour accéder à la fonte de manière normale.

Ça n'est pas d'un intéret débordant, mais lorsque l'on parle d'un programme, il est bon d'utiliser son logo de manière convenable.

2.37 Dessiner l'heure : le package `clock`

Ce petit package, et la fonte qui va avec, que nous devons à Oliver Corff [49], permet de dessiner des horloges à aiguilles indiquant une heure choisie.

Le package ne prend pas d'options. Tous les règlages se font dans le document. Il existe 4 styles d'horloge, que l'on peut sélectionner avec la commande `\Clockstyle` (c'est un compteur TEX), qui peuvent avoir un bord ou pas, fonction de l'usage de `\ClockFrametrue` (active la présence du bord) ou de `\ClockFramefalse` (désactive le bord). Le tableau 2.26 page suivante présente les 8 horloges possibles, indiquant toutes 14h35.

Style	**Avec bord**	**Sans bord**
0		
1		
2		
3		

TAB. 2.26: Les huit variétés d'horloges de `clock`

`\clock{`⟨*heures*⟩`}{`⟨*minutes*⟩`}`

La commande qui produit l'horloge elle-même est `\clock`. Tout cela s'utilise par exemple comme ça :

```
\ClockStyle=1
\ClockFrametrue
\clock{14}{35}
```

pour produire : .

La commande `\clocktime` indique l'heure de la compilation du document, soit ici :

2.38 Écrans à cristaux liquides : le package `lcd`

Ce package, que nous devons à Mike KAUFMANN [96], permet d'écrire comme un écran à cristaux liquides.

Le rendu de ce package est assez réaliste, mais son utilisation est franchement pénible. On doit probablement pouvoir faire mieux, mais pour le moment, il faut se contenter de ça.

2.38.1 Utilisation de base

Le principe est simple, on tape, par exemple[28] :

```
\begin{center}
%                  1         2         3         4
%         1234567890123456789012345678901234567890
\LCD{3}{40}+*****************************************+
           +*              EXEMPLE                  *+
           +*****************************************+
\end{center}
```

pour produire :

[28] J'ai laissé le commentaire qui m'a permit de compter les colonnes, bien entendu, TEX n'en tient pas compte : c'est un commentaire.

```
****************************************
*               EXEMPLE                *
****************************************
```

On ne peut pas rentrer ce qu'on veut comme texte, seuls certains symboles sont définis par le package. Cependant, certains symboles complexes peuvent être saisis, par exemple :

```
\begin{center}
%                          1
%           123     456789012345
\LCD{1}{15}+1 {euro} = 6,55957 F+
\end{center}
```

produit :

```
1 € = 6,55957 F
```

Attention : on ne peut pas rentrer un symbole complexe, comme `{euro}`, en dernier, il ne serait pas correctement traité. Si tu as besoin de le mettre en dernier, ajoute un espace derrière. Tu peux même mettre l'espace *en trop* : le package ne vérifie pas le nombre de caractères que tu tapes, l'espace en trop sera donc affiché en dépassant du cadre, ce qui ne se verra pas.

La commande `\textLCD` fonctionne globalement comme `\LCD`, mais permet d'écrire directement dans le texte, ainsi, à partir de ce source là :

```
... le texte, \textLCD{5}+ainsi+, \`a...
```

Ces deux commandes, `\LCD` et `\textLCD` sont fragiles. Elles fonctionnent un peu sur le même mode que `\verb`, et donc ne peuvent pas être utilisées n'importe où, en particulier pas dans les titres de section ou de chapitre, ou dans les tableaux, sauf si leur argument ne contient que des caractères neutres (par de %, de $ ou de !, par exemple).

La commande `\LCDnoframe` permet de désactiver le cadre produit par `\LCD`. Au contraire, `\LCDframe` permet de le réactiver.

Très peu de caractères sont prédéfinis par le package. En gros, ça se limite aux lettres non-accentuées (minuscules et majuscules), à la ponctuation usuelle, aux trèmas classiques de l'allemand (l'auteur est allemand), et aux symboles complexes décrits par le tableau 2.27 page suivante.

2.38.2 Utilisation de la couleur

Ce package interragit assez intelligemment avec la gestion des couleurs du package `color`, via la commande `\LCDcolors`.

`\LCDcolors[`⟨*cadre*⟩`]{`⟨*texte*⟩`}{`⟨*fond*⟩`}`

Le premier paramètre (optionnel) indique s'il doit y avoir un cadre et de quel couleur. Le deuxième paramètre indique la couleur du texte, et le troisième indique la couleur de fond. Par exemple, pour écrire en blanc sur noir avec un cadre rouge :

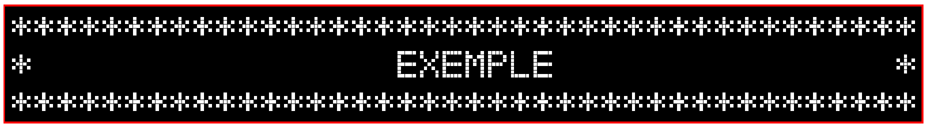
```
\LCDcolors[red]{white}{black}
\begin{center}
%                  1         2         3         4
%           1234567890123456789012345678901234567890
```

Nom	Symbole
{clock}	◷
{rarrow}	→
{larrow}	←
{Omega}	Ω
{Sigma}	Σ
{pi}	π
{sqrt}	√
{rect}	□
{fcur}	█
{lb}	{
{rb}	}
{alpha}	α

TAB. 2.27: Symboles prévus par le package `lcd`

```
\LCD{3}{40}+****************************************+
           +*               EXEMPLE               *+
           +****************************************+
\end{center}
```

qui produira[29] :



2.38.3 Définir de nouveaux caractères

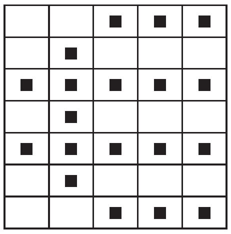
La commande `\DefineLCDchar` permet de définir un nouveau symbole, qu'il soit complexe ou pas. C'est comme ça que j'ai définit le symbole euro :

```
\DefineLCDchar{euro}{00111010001111101000111110100000111}
```

Le premier paramètre de la commande `\DefineLCDchar` est, bien entendu, le symbole que l'on définit. Le deuxième paramètre est une suite de 35 chiffres, des 0 et des 1, qui décrivent le caractère à produire. Il y a, en fait, 7 groupes de 5 chiffres, chaque groupe décrivant une ligne, par exemple d'après le tableau ci-dessous :

---

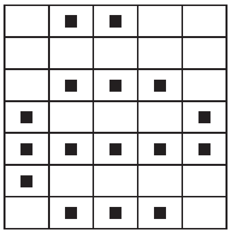
[29]Pour les gens qui n'ont pas la couleur, ou pour les daltoniens, la couleur du cadre est définie par le premier argument, optionnel, et est donc rouge dans l'exemple, la couleur du texte est définie par le deuxième argument, obligatoire (blanc dans l'exemple), et la couleur de fond est définie par le troisième argument, obligatoire (noir, dans l'exemple).



```
⟶ 00111
⟶ 01000
⟶ 11111
⟶ 01000
⟶ 11111
⟶ 01000
⟶ 00111
↓
00111 01000 11111 01000 11111 01000 00111
↓
00111010001111101000111110100000111
```

De la même manière, je me suis définti un « é » en appelant :

```
\DefineLCDchar{eacute}{01100000000111010001111111000001110}
```



```
⟶ 01100
⟶ 00000
⟶ 01110
⟶ 10001
⟶ 11111
⟶ 10000
⟶ 01110
↓
01100 00000 01110 10001 11111 10000 01110
↓
01100000000111010001111111000001110
```

2.38.4 Déception

Dans l'ensemble, ce package semble une idée amusante, mais laisse franchement à désirer en matière de confort d'utilisation ou en robustesse. Tout vient du fait que cette « fonte » n'en est pas une. Les macros `\textLCD` et `\LCD` lisent leur argument et le dessinent lettre à lettre. Si une vraie fonte était définie, tous ces problèmes disparaitraient.

2.39 Touches de clavier : le package `keystroke`

Ce package, que nous devons à Rolf Niepraschk (non documenté), permet de dessiner des touches de clavier. Par exemple :

```
\Ctrl-\Alt-\Del\ est la commande...
```

produira : Ctrl - Alt - Suppr est la commande...

Ce package n'utilise pas une fonte, mais plutôt des images qu'il place en arrière plan, avec le texte en premier plan. Il se base sur le package `graphics` et peut donc produire des résultat aussi bien en DVI qu'en PDF.

Les tableaux 2.28 et 2.29 page suivante présentent les commandes définies par ce package pour les touches classiques. Les commandes du premier tableau sont affectées par l'option (ça n'a pas de sens d'en mettre plus d'une, ce qui est bien regretable) passée au package. Les options permettent de modifier la langue de ces commandes. Les options suivantes sont définies : english (par défaut), american (évalent à english), german, ngerman, french, et spanish. C'est très dommage qu'il ne sache pas s'adapter aux changements de langue de `babel`.

Commande	Symbole
\Enter	Entrée
\Del	Suppr
\Ins	Ins
\Esc	Esc
\Shift	Shift ⇑
\Ctrl	Ctrl
\Home	Home
\End	Fin
\PgUp	Pagina ↑
\PgDown	Pagina ↓
\PrtSc	PrtSc
\Scroll	Scroll
\Break	Break

TAB. 2.28: Touches de clavier, selon la langue (package `keystroke`)

Commande	Symbole
\Spacebar	
\Return	↩
\BSpace	⟵
\Tab	⇆
\Alt	Alt
\AltGr	AltGr
\NumLock	Num
\UArrow	↑
\DArrow	↓
\LArrow	←
\RArrow	→

TAB. 2.29: Touches de clavier, indépendantes de la langue (package `keystroke`)

Une commande générale permet de faire un peu ce qu'on veut : `\keystroke`, qui s'utilise comme ça :

```
\keystroke{Pause} \keystroke{2}
```

et produit : $\boxed{\text{Pause}}$ $\boxed{2}$.

Du coup, en se donnant un peu de mal, on peut définir ces deux macros[30] :

```
\newcommand{\monPgDown}{%
  \keystroke{{%
    \setbox123=\hbox{$=$}\relax
    \hbox{\ \hskip .5\wd123\relax
          \clap{$\downarrow$}%
          \hskip -.5\wd123\relax
          \raise 1pt\box123\relax
          \ }}}}
\newcommand{\monPgUp}{%
  \keystroke{{%
    \setbox123=\hbox{$=$}\relax
    \hbox{\ \hskip .5\wd123\relax
          \clap{$\uparrow$}%
          \hskip -.5\wd123\relax
          \lower 1pt\box123\relax
          \ }}}}
\monPgUp et \monPgDown
```

qui produisent : ⇞ et ⇟ (nettement mieux que les fautes de traduction proposées au tableau 2.28 page précédente).

Un micro-package (`fixks`) reprenant ces macros est disponible à l'URL :
`http://jmpl.fr.eu.org/JMPL`

2.40 Jouer aux dés : le package `epsdice`

Ce package, que nous devons à Thomas Heim [83], permet de dessiner des dés tout simples. Il fournit une unique commande, `\epsdice`, qui s'utilise comme ça :

```
\epsdice{1}
\epsdice{5}
```

pour produire : ⚀ ⚄. Cette commande s'adapte à la taille en cours, elle marche donc dans les titres, les notes de pied de page, etc.

[30] Pour une explication sur l'utilisation de `\newcommand`, `\hbox`, `\raise`, `\lower`, `\hskip`, et `\wd`, reporte-toi au chapitre neuvième page 571.

Pour ce qui concernes la commande `\clap`, ce n'est pas une commande de base de LaTeX. Par contre il existe deux commandes `\rlap` et `\llap` que tous les programmeurs TeX connaissent. Du coup, on a souvent tendance à utiliser la commande `\clap` tellement il serait logique qu'elle existe. C'est à tel point qu'on la retrouve régulièrement dans un grand nombre de package. Le hasard fait que la commande telle que je l'utilise ici est définie par le package `hhflxbox`.

Si tu devais définir toi-même cette commande, voilà sa définition (similaire aux définitions de `\rlap` et `\llap`) : `\def\clap#1{\hbox to \z@{\hss #1\hss}}`

Ces macros (`\monPgUp` et `\monPgDown`) font des choses relativement simple, en fait : elles centrent une flèche sur un =, en décalant légèrement le = vers le haut ou vers le bas. Le mode de centrage est légèrement magique, enfin, délicat à comprendre pour qui n'a pas lu le TeXbook ([108] ou [109]). L'utilisation arbitraire du registre 123 ne pose pas de problème parce qu'elle est locale à un groupe bien déterminé[31].

[31] Si tout ça te semble écrit en chinois, c'est pas grave. Il suffit d'utiliser les macros sans les comprendre, si tu es fainéant, ou d'aller lire les références que je cite, si tu es courageux.

2.41 Du braille en anglais : le package `braille`

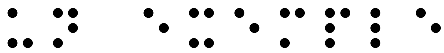
Ce package, écrit par William Park et Dominique Unruh [169, 170], permet de typographier du braille en *anglais*.

Tu te demandes peut-être en quoi la langue compte pour le Braille, tout comme je me suis posé la question moi-même, hé bien c'est très simple : le braille peut soit être utilisé lettre à lettre, en transcodage direct de la langue de l'auteur, auquel cas la langue du texte n'a aucune importance, du moment qu'on reste sur un alphabet latin ; ou bien il peut être codé de manière nettement plus dense en utilisant des tas d'abréviations qui, elles, dépendent de la langue.

Ça correspond a deux niveaux d'usages du braille, le niveau 1 étant litéralement lettre à lettre, et le niveau 2 étant plus condensé.

Je ne m'étendrai pas sur le détail du niveau 2, d'abord parce que la documentation du package est claire, ensuite parce que ça ne traite que de l'anglais. Le package serait assez simple à adapter pour pouvoir traiter plusieurs langues.

Rapidement, un exemple. On obtient :

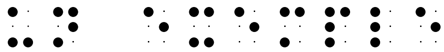


en tapant :

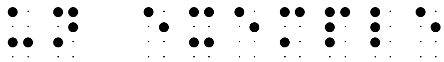
```
\braille{un exemple}
```

2.41.1 Les options du package

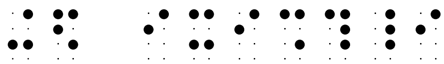
L'option puttinydots demande à afficher de petits points aux endroits qui ne sont *pas* percés, en plus des gros points aux endroits percés. Elle peut être activée à tout moment par appel de `\brailleputtinydotstrue` et désactivée par `\brailleputtinydotsfalse`. En activant cette option, l'exemple précédent devient :



Une option, 8dots, permet de passer en braille 8 points au lieu du classique 6 points (je n'ai pas la moindre idée de ce à quoi ça peut bien servir). Elle peut être activée à tout moment par appel de `\brailleeightdotstrue` et désactivée par `\brailleeightdotsfalse`. En activant cette option, l'exemple précédent devient :



L'option mirror, permet d'imprimer les caractères en « miroir » pour pouvoir percer la feuille avec une épingle et qu'elle devienne véritablement lisible par un aveugle[32]. Elle peut être activée à tout moment par appel de `\braillemirrortrue` et désactivée par `\braillemirrorfalse`. En activant localement cette option, l'exemple précédent devient :

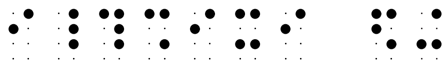


[32] Si j'ai bien compris le fonctionnement du braille, ce qui n'est pas sur, il faudrait également inverser les caractères, *i.e.* écrire de gauche à droite, ce que plusieurs extensions de TEX permettent, ainsi, après perçage de la feuille, un aveugle peut la lire au verso.

En combinant ça avec une extension capable d'écrire de droite à gauche, on obtient un texte entièrement à l'envers. Un tel texte, s'il est percé convenablement, pourra être lu par un aveugle, ce qui est tout de même le but du package. Voici un exemple de ce que ça peut donner[33] :

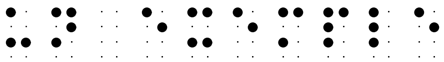
```
\begin{center}
\textdir TRT
\braille{un exemple}
\end{center}
```

produit ce résultat-là :



L'option compact permet de ne *pas* charger la commande `\braille` ni toutes les tables d'abréviations, ce qui ne laisse comme seule commande utile que la commande `\braillebox`.

Enfin, l'option useemptybox demande au package d'utiliser un caractère braille vide (*i.e.* sans gros points) à la place des espaces. La seule différence visible doit donc être que le package dessinera 6 (ou 8, selon le cas) petits points à la place du vide si l'option puttinydots est active. L'option useemptybox peut être activée a tout moment par appel de `\brailleuseemptyboxtrue` et désactivée par `\brailleuseemptyboxfalse`. Enfin, toujours le même exemple, mais avec cette option :



2.41.2 L'écriture d'un caractère quelconque

Les commande `\braillebox` se cache derrière chacun des caratères produits pas la commande `\braille`.

Chacun des points est numéroté selon cette grille :

1	4
2	5
3	6
7	8

La commande `\braillebox` prend en argument la liste des points à percer. Ainsi, ce source :

```
\begin{center}
\braillebox{1}
\braillebox{12}
\braillebox{123}
\braillebox{1234}
```

[33]Ici, j'ai choisi d'utiliser les primitives d'Ω pour faire ce changement de direction. Ω est une version modifiée de TEX qui apporte un grand nombre d'extensions, dont, entre autres, le fait de pouvoir écrire de droite à gauche de manière native. Bien entendu d'autres extensions qu'Ω peuvent être utilisées, dont certains packages pour LATEX, par exemple ceux destinés à écrire de l'hébreu ou de l'arabe.

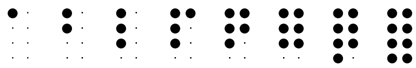
Dans le cas de l'utilisation avec la primitive `\textdir` d'Ω, il n'est pas utile de positionner l'option `\braillemirrortrue`, parce que le package `braille` produit chaque lettre sous forme de deux caractères, un pour chaque colonne de points. Comme Ω va inverser tout ça, il va non seulement inverser les lettres, mais aussi les colonnes, produisant le résultat escompté.

Commande	**Symbole**
`\manhpennib`	
`\manvpennib`	
`\mantiltpennib`	
`\mancube`	
`\manimpossiblecube`	
`\manquadrifolium`	
`\manrotatedquadrifolium`	
`\manstar`	
`\manerrarrow`	
`\dbend`	
`\lhdbend`	
`\reversedvideodbend`	

TAB. 2.30: Symboles divers (package `manfnt`)

```
\braillebox{12345}
\braillebox{123456}
\braillebox{1234567}
\braillebox{12345678}
\end{center}
```

produira ce résultat :



2.42 Symboles pour les fans : le package `manfnt`

Ce petit package, que nous devons à Axel KIELHORN [99], n'est pas d'un intérêt débordant, mais il permet l'accès à un certain nombre de symboles que les fans de TEX connaissent bien : ce sont les symboles que Donald E. KNUTH a dessiné pour utiliser dans le TEXbook et le METAFONTbook. Par exemple le symbole au début de ce paragraphe, qu'il utilise pour indiquer les passages compliqués du livre, qu'il est plus prudent de ne lire qu'en deuxième ou en troisième lecture.

Pour utiliser le symbole en début de paragraphe, j'ai fait dans le brutal : je me suis définit une commande `\Dbend` qui fait les petits calculs utiles pour réserver 8 millimètres sur deux lignes pour placer le symbole. Voilà comment j'ai défini ma commande et comment je m'en suis servi[34] :

```
\newcommand\Dbend{%
  {\advance \linewidth by -8mm\relax
   \xdef\lignecourte{\the\linewidth}}
```

[34]Pour l'usage de la commande `\newcommand`, reporte toi au chapitre neuvième page 571 qui traite de la programmation. Pour ce qui est de l'usage de `\parshape`, il te faudra te reporter au TEXbook [108, 109].

```
\noindent
\parshape 3 8mm \lignecourte
            8mm \lignecourte
    0pt \linewidth
\llap{\hbox to 8mm{\dbend\hfill}}

\Dbend Ce petit package...
```

La fonte où on trouve ce symbole est très largement diffusée, parce qu'elle fait historiquement partie des fontes de base de TeX, c'est en particulier celle qui contient les lettres pour le logo METAFONT. Elle n'offre cependant qu'un intérêt limité, la majorité des symboles qui s'y trouvent sont des exemples pour le METAFONTbook, et ne sont donc pas intéressant en tant que symboles, mais comme résultat d'un exemple.

Les quelques symboles qui pourraient être utiles sont repris dans le tableau 2.30 page précédente.

2.43 Faire de jolis cadres : le package `niceframe`

Ce petit package, que nous devons à Marcus OHLHAUT [167], permet de dessiner quelques cadres amusants à partir des fontes dingbats.

`\niceframe[`⟨*largeur*⟩`]{`⟨*texte*⟩`}`

`\curlyframe[`⟨*largeur*⟩`]{`⟨*texte*⟩`}`

`\artdecoframe[`⟨*largeur*⟩`]{`⟨*texte*⟩`}`

Si on se limite à un usage simple, ce package propose trois commandes, qui ont toutes les trois la même syntaxe : `\niceframe`, `\curlyframe` et `\artdecoframe`. Ces commandes prennent en premier paramètre, optionnel donc entre crochets, la largeur du cadre souhaité, et en second paramètre le texte à encadrer. Par défaut, le cadre fera la largeur du texte en cours (la largeur de la page, ou la largeur de la zone de texte en cours).

Un petit exemple :

Ceci est un exemple un petit peu long du package `niceframe`. On utilise ici tous les règlages par défaut, pour voir ce que ça donne.

est obtenu à partir de :

```
\niceframe{Ceci est un exemple un petit peu long du package
\textpak{niceframe}. On utilise ici tous les r\'eglages par
d\'efaut, pour voir ce que \c{c}a donne.}
```

Voilà ce que produira la
commande `\curlyframe` sur une
largeur de 7 cm.

Voici un exemple d'appel à la commande
`\artdecoframe`, en lui précisant une largeur de
10 cm.

`\generalframe{`⟨*haut-gauche*⟩`}{`⟨*haut*⟩`}{`⟨*haut-droite*⟩`}{`⟨*gauche*⟩`}{`⟨*droite*⟩`}` `{`⟨*bas-gauche*⟩`}{`⟨*bas*⟩`}{`⟨*bas-droite*⟩`}{`⟨*texte*⟩`}`

Si on ne se limite pas à un usage simple, la commande `\generalframe` permet de construire des cadres à partir d'une fonte contenant 8 caractères bien choisis. Cette commande prend 9 paramètres : les 8 caractères en question, et le texte à encadrer. Plutot que de te faire une longue explication sur le rôle de chacun des paramètres, ce petit essai devrait te suffir à comprendre :

```
\generalframe {1}{2}{3}{4}{5}{6}{7}{8}{Ceci est un texte \'a
  encadrer, pour voir ce que \c{c}a donne. Le r\'esultat est
  tr\'es laid, mais montre bien le r\^ole des 9 param\'etres.}
```

produira :

122222222222222222222222222222222222222222222222222222222222222222222222223
4 5
4 Ceci est un texte à encadrer, pour voir ce que ça donne. Le résultat est très laid, mais 5
4 5
4 montre bien le rôle des 9 paramètres. 5
4 5
677777777777777777777777777777777777777777777777777777777777777777777777778

Dans la documentation du package, l'auteur donne quelques exemples basés sur les fontes `umranda` et `umrandb` de Alexander Schrell, fontes qui sont distribuées avec `niceframe`. La fonte `umrandb`, tout particulièrement, permet de dessiner de nombreuses variétés de cadres. Plutot que de me limiter aux trois exemples qui sont dans [167], j'ai repris toutes les combinaisons possibles dans un micro-package, `myniceframe`. Les figures 2.3 et 2.4 te montrent des exemples des commandes proposées par le package `myniceframe`. Ce package est disponible en ligne à l'URL suivante :
`http://jmpl.fr.eu.org/JMPL/myniceframe.sty`

FIG. 2.3: Cadres du package `myniceframe`

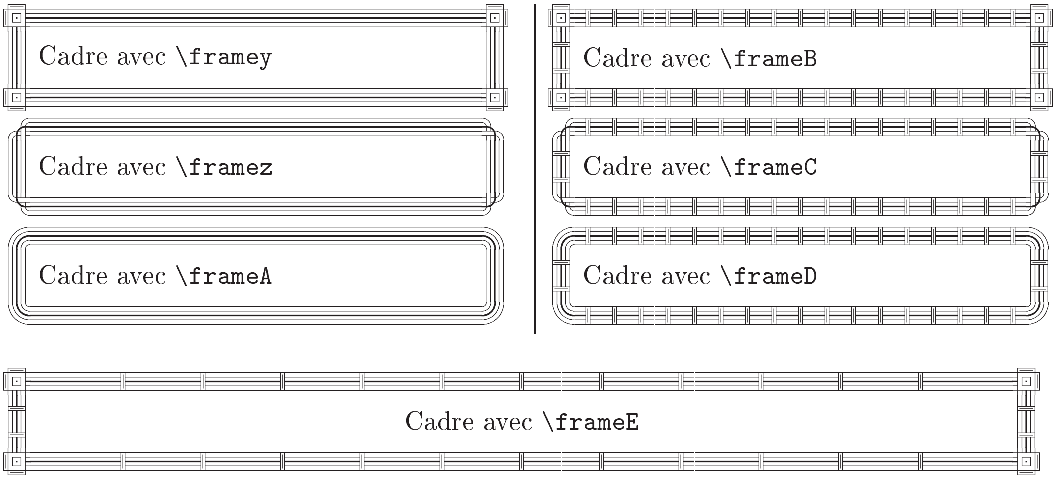


FIG. 2.4: Cadres du package `myniceframe`

2.44 Du souligné qui marche mieux : le package `ulem`

Le package `ulem` que nous devons à Donald ARSENEAU [6] est un joli reste de LaTeX 2.09 qui a été sommairement remis à jour pour LaTeX 2ε. Il permet de gérer de manière plus poussée les soulignements. Je sais, c'est pas en lien étroit avec les problèmes de changement de fontes, mais je voyais pas où le mettre ailleurs.

Ce package remplace les deux commandes `\em` et `\emph` de manière parfaitement transparente pour qu'au lieu de mettre du texte en italique elles le soulignent. C'est là sa première raison d'être, bien que, de mon point de vue, ce soit plutôt un effet secondaire désagréable. Je précise donc tout de suite que ce package admet plusieurs options, en particulier :

normalem laisse les deux commandes `\em` et `\emph` inchangées,

ULforem fait le contraire,

normalbf indique de ne pas toucher aux commandes `\bfseries` et `\textbf`,

UWforbf indique d'utiliser un soulignement ondulé à la place du gras.

Pour ma part, histoire de ne pas perturber mes petites habitudes avec LaTeX j'aurais tendance à préciser les deux options normalem et normalbf. Mais il parait que certains éditeurs attendent du souligné à la place de l'italique, et dans ce cas précis, ça peut être pratique de ne pas avoir à reprendre tout le document.

Une des fonctionnalités intéressantes apportée par ce package est le fait que les commandes qu'il définit pour le soulignement sont capables de gérer convenablement les changement de lignes dans les parties soulignées, ce qui n'est pas le cas pour la commande `\underline` de LaTeX.

Par contre, certaines restrictions sont annoncées dans la « documentation[35] » du package. En effet, le changement de ligne est géré automatiquement, les commandes de soulignement définies par `ulem` sont incapables d'utiliser les césures, par contre elles sont capables de tenir compte de la commande `\-` qui indique explicitement un point de césure possible (voir à ce sujet le paragraphe 4.23.2 page 291) ; de même ces commandes savent tenir compte

[35] On peut difficilement parler de documentation puisqu'il s'agit du fichier source du package, cependant, ce fichier contient toute une partie explicative sur le fonctionnement du package, à la fin.

de commandes comme `\newline` ou `\linebreak`. Cependant, elles seront très fortement perturbées par les appels de commandes et par les accolades. Ainsi, s'il te venait la fantaisie de mettre un niveau d'accolades de trop lors de l'appel d'une de ces commandes, alors toutes les possibilités de passage à la ligne disparaitraient. De même, le fait de mettre une partie du texte souligné en gras[36] posera problème.

Bien, rentrons un petit peu dans le vif du sujet. Ce package crée, pour l'utilisateur de base, 5 commandes :

`\uline` pour utiliser du soulignement normal et qui sait se propager convenablement sur les changements de lignes.

`\uuline` le même pour le double soulignement.

`\uwave` le même avec un soulignement ondulé.

`\sout` pour rayer du texte ~~comme ça~~.

`\xout` pour rayer du texte comme ça.

Je ne pense pas que cela nécessite beaucoup plus d'explication pour un usage simple, le reste sera donc consacré à un usage avancé, voire très avancé.

Donald Arseneau donne deux exemples intéressants pour un usage avancé de son package, je vais les reprendre tels quels et les commenter ensuite :

```
L'exemple simple:

\noindent 'Twas {\em brillig\/} and the {\em slithy~toves\/}
did {\em gyre\/} and {\em gim\-ble\/} in the {\em wabe,\\[2pt] }
All {\em mim\-sey\/} were the {\em boro\-goves\/} and
the {\em mome raths outgrabe}.

L'exemple compliqu\'e:

No, I did {\em not} act in the movie {\em \emph{The} % <<<<<<< Nested
\emph{Persecu}\-\emph{tion} \emph{and} \emph{Assassination} \emph{of}
\emph{Jean-Paul} \emph{Marat}, as per\-formed by the Inmates
of the Asylum of Charenton under the Direc\-tion of the
Marquis de~Sade!} But I {\em did} see it.
```

Ce qui produit le résultat suivant :

L'exemple simple :

'Twas brillig and the slithy toves did gyre and gimble in the wabe,
All mimsey were the borogoves and the mome raths outgrabe.

L'exemple compliqué :

No, I did not act in the movie The Persecution and Assassination of Jean-Paul Marat, as performed by the Inmates of the Asylum of Charenton under the Direction of the Marquis de Sade ! But I did see it.

Dans le cas de l'exemple compliqué, on relèvera que l'imbrication de passages soulignés[37] n'est pas simple à obtenir. En particulier, les espaces sont laissées en dehors de l'imbrication pour leur permettre de changer de taille[38] et pour autoriser le changement de ligne. De

[36] Par exemple, parce que, en italique ou en plus petit ou en plus grand, ou ... ça pose les mêmes problèmes bien entendu.
[37] Ou, plus précisément, l'appel de `\emph` dans `\em`, puisque cet exemple utilise le fait de remplacer `\em` par du souligné.
[38] Pour pouvoir faire la justification.

Commande	**Exemple**
`\so`	I n t e r l e t t r a g e
`\caps`	CAPITALES
`\ul`	souligné
`\st`	~~barré~~
`\hl`	surligné

TAB. 2.31: Exemples d'utilisation des commandes du package `soul`

même, le `\-` est laissé en dehors de l'imbrication pour permettre à la césure de se faire en cas de besoin.

L'exemple simple permet de tester une bonne partie des commandes de changement de ligne et de gestion de l'espacement que tolère ce package, comme par exemple le `~` ou le `\\`.

Enfin, pour conclure, deux commandes fournies par ce package et pour lesquelles je ne donne que des exemples (à manipuler avec précaution, cela va de soit) :

```
\useunder{\uline}{\em}{\emph}
et
\normalem
```

Ces deux commandes permettent d'utiliser momentanément du soulignement à la place de l'italique produite par `\em` et `\emph`.

2.45 Du souligné qui marche mieux : le package `soul`

Ce package, que nous devons à Melchior FRANZ [68], offre essentiellement deux fonctionnalités : l'interlettrage (*spacing out*, en anglais) et le souligné (*under line*, en anglais), d'où son nom (`soul`[39]).

Bon, on va commencer par rappeler deux choses importantes : l'interlettrage, ça ne se fait normalement pas en dehors de quelques usages très précis (en allemand, par exemple, c'est bien vu quand on écrit un titre tout en capitales de l'écrire avec un peu d'interlettrage) ; quand au souligné, c'est considéré comme *très mal*. Donc, tout cela n'est à utiliser que si tu te retrouves avec des cas extrêmes : un chef qui insiste, un directeur de thèse qui refuse d'entendre raison, etc.

2.45.1 Usage de base

Le package `soul` propose essentiellement 5 commandes intéressantes. La commande `\so`[40] fait de l'interlettrage, la commande `\caps`[41] fait des capitales légèrement espacées (il parait que les allemands aiment ça), la commande `\ul`[42] produit du souligné, la commande `\st`[43] produit du barré, et `\hl`[44] produit du surligné.

Dans le résultat produit par le package, plusieurs choses sont paramètrables. Par exemple la position du trait du souligné qui peut être plus ou moins bas par rapport à la ligne, ou la

[39]Le mot *soul* en anglais désigne l'âme, on aurait pu y chercher un cause mystique, ce n'est pas le cas, c'est juste un mauvais jeu de mots.
[40]`\so` pour *spacing out*.
[41]`\caps` pour capitales.
[42]`\ul` pour *under line*.
[43]`\st` pour *striking*.
[44]`\hl` pour *highlight*.

couleur de surligné.

`\setul{⟨profondeur⟩}{⟨épaisseur⟩}`

Cette commande permet de fixer à quelle profondeur sous la ligne porteuse, et de quelle épaisseur, doit être le souligné. Ainsi, après cet appel, le souligné sera moins bas que par défaut, et sera un peu plus épais (pas forcément une très bonne idée) :

```
\ul{avant}
\setul{1pt}{1pt}
\ul{apr\`es}.
```

produit ça : avant après. L'épaisseur du souligné par défaut est de 0,4 pt, comme la majorité des filets que LATEX trace.

La commande `\resetul` permet de revenir aux valeurs par défaut du package, pour annuler tes changements.

`\setuldepth{⟨texte⟩}`

Le plus souvent, on souhaite fixer la profondeur du souligné non pas de manière absolue, comme ce que propose la commande précédente, mais en accord avec la fonte utilisée : pour une fonte dont les lettres descendantes sont plus profondes, on pourra vouloir rabaisser le souligné. C'est ce que propose la commande `\setuldepth` : fixer le souligné 1 point en dessous de la lettre la plus descendante du texte qui lui est passé. Si tu passes un texte vide, alors la commande va choisir comme référence le caractère le plus profond de la fonte en cours[45]. La première ligne te permettra de demander un souligné qui coupe les descendantes, et la deuxième un souligné qui passe en dessous à tous les coups (très en-dessous), parce qu'en général, dans une fonte, les parenthèses sont prévues pour descendre en dessous des lettres descendantes :

```
\resetul
\ul{Normal (plage)},
\setuldepth{a}
\ul{pattes g\'en\'eralement coup\'ees},
\setuldepth{()}
\ul{bas, (tr\`es) bas: plage}.
```

produira ça : Normal (plage), pattes généralement coupées, bas, (très) bas : plage.

`\setulcolor{⟨couleur⟩}`

La commande `\setulcolor` permet de changer la couleur du trait de souligné. Il ne faut pas en abuser, ça peut produire des résultats du dernier mauvais goût. C'est basé sur le package `color` pour la couleur (voir le paragraphe **??** page ?? pour une description de ce package). Pour produire cet exemple de mauvais goût ultime, il suffit de faire appel à[46] :

```
\setulcolor{red}%
```

[45]En fait, c'est légèrement plus subtil. Le package va chercher, parmi les 256 caractères, lesquels sont indiqués comme étant des lettres, et va utiliser celles-ci dans sa recherche de la plus profonde. Si ta fonte comporte un symbole qui n'est pas une lettre, par exemple une ligature comme fi, qui est plus profond que n'importe quelle lettre, alors le package ne le verra pas. De même, dans les extensions de TEX qui prévoient plus de 256 caractères (c'est le cas d'Ω, qui fonctionne en Unicode et prévoit un nombre de symboles qui s'exprimerait plutôt en millions), le package se limitera aux 256 premiers symboles classés comme des lettres : il y a fort à parier que ce sera l'alphabet latin.

[46]Pour ceux qui ont une version en noir et blanc (ou qui ne voient pas bien les couleurs), le texte est écrit dans un vert bien claquant, et est souligné en rouge. Le résultat est laid, très laid. Il ne faut pas faire ça. C'est mal.

```
\textcolor{green}{\ul{Pour produi...}}
```

`\setstcolor{⟨couleur⟩}`

`\sethlcolor{⟨couleur⟩}`

De la même manière, on peut modifier la couleur du surligné[47] avec la commande `\sethlcolor` et celle du barré avec `\setstcolor`. Si l'idée (attroce) de combiner `\hl` et `\ul` te venait, sâche que ça ne marche pas.

2.45.2 Comprendre la mécanique

Ce que fait ce package peut sembler très simple, parce que le résultat est très simple à comprendre, mais la façon de faire ça avec LaTeX n'est vraiment pas évidente. La macro `\ul`[48] fait subir un traitement très complexe à son paramètre : dans un premier temps, elle le décompose en briques élémentaires (dans le cas simple, ce sont des lettres). Chaque brique élémentaire est analysée et remplacée par elle-même (suivie d'un blanc le plus souvent) après avoir subi un traitement particulier.

En procédant comme ça, et en supposant que `\X` soit le traitement à faire subir à chaque brique élémentaire, un appel à `\ul{Essai}` devient quelques chose comme :

```
\X{E}\X{s}\X{s}\X{a}\X{i}
```

où la macro `\X` fait le traitement (par exemple souligner) et ajoute (ou pas) un peu de blanc entre les lettres (pour l'interlettrage). Le résultat de tout ce traitement, pour LaTeX, n'est pas un mot : pour LaTeX, un mot, c'est une suite pure et simple de lettres, le premier machin qui n'est pas une lettre fait que le mot s'arrête. Or LaTeX ne cherche des césures que dans ce qu'il pense être des mots. Du coup, si on veut que les césures puissent avoir lieu, il faut que la macro `\ul` cherche, dans le texte d'origine, les éventuels points de coupure, et les ré-indique explicitement à LaTeX dans le résultat produit.

À chaque fois que la macro `\ul` pense avoir obtenu un mot à force d'accumuler des briques, elle va chercher tous les points de césure possibles en demandant à LaTeX de le faire (ça fait que le package connaît autant de langues que ce que connaît LaTeX), puis à chaque point de coupure possible, il sera indiqué à LaTeX sous forme explicite dans le résultat[49] :

```
\X{E}\X{s}\-\X{s}\X{a}\X{i}
```

Une fois ce principe de fonctionnement compris, il parait quasiment évident que de très nombreuses choses ne vont pas marcher. Par exemple, un appel aussi simple que :

```
\ul{Ah mais \textbf{non}!}
```

va poser, *a priori*, des tas de problèmes. Comme ça, spontanément, on voit deux sources de problèmes : comment sera traité `\textbf`, et comment seront traitées les deux accolades. Dans un cas simple, on va arriver à une construction comme celle-là :

```
...\X{m}\X{a}\X{i}\X{s}\X{ }\X{\textbf}\X{{non}}\X{!}
```

[47] Amusant, ce nom, parce que, techniquement, le rectangle de couleur est *sous* le texte, mais souligné, c'était déjà pris.

[48] Dans l'explication je parlerais de `\ul`, mais en fait les macros `\ul`, `\hl`, etc., sont toutes basées sur le même moteur de traitement. Elles correspondent à des paramètrages différents d'une macro plus générique.

[49] La réalité de ce qui se passe est plus complexe, par exemple la façon de demander à LaTeX les points de coupure n'est absolument pas simple. Mon but ici n'est pas de t'expliquer comment ré-écrire le package, mais juste de te présenter le principe du fonctionnement, et ce qu'on peut du coup en attendre.

ce qui va poser problème : `\textbf` va prendre un argument pour le mettre en gras. Dieu[50] seul (et l'auteur du package, peut-être) sait ce qui se trouvera derrière `\textbf` à ce moment là, mais il y a fort à parier que ce ne sera pas ce qu'on voudra. Du coup, le package connaît un bon paquet de macros et sait comment les traiter. Dans le cas de `\textbf`, le résultat sera quelque chose comme ça :

```
...\X{m}\X{a}\X{i}\X{s}\X{ }\textbf{\X{n}\X{o}\X{n}}\X{!}
```

qui est ce qu'on voulait.

Le problème se posera aussi quand on saisi les accents à l'ancienne (comme je le fais moi, dans les exemples), avec des constructions comme `\'E` où il ne faut pas séparer les deux éléments constituant la lettre accentuée. En fait, le problème va se poser pour à peu près tout ce qui n'est pas une lettre.

Pour s'en convaincre il suffit de regarder cet exemple : efficace devient efficace en souligné (la différence est dans la ligature ffi qui disparait à cause du point de césure, puisqu'elle est transformée en `f\-fi`), et e f f i c a c e avec de l'interlettrage (là aussi, la ligature disparaît complètement à cause des blancs ajoutés qui empêchent LaTeX de faire les ligatures).

Une particularité de saisie est prévue : si tu veux que certains caractères soient traités comme une seule brique, il suffit de les regrouper par des accolades. Cependant, dans ce groupe *il ne doit pas y avoir de point de césure*. Si tu veux regrouper des caractères dans lesquels il y a un point de césure, il faudra les mettre dans une `\mbox`. De plus, la commande `\<` indique, quand on fait de l'interlettrage, de ne pas ajouter de blanc entre ces deux briques, par exemple pour les nombres, tu peux les protéger comme `2\<5` ou comme `{2005}`.

`\setuloverlap{⟨longueur⟩}`

Quand le package fait du soulignement, ou assimilé (c'est-à-dire les commandes `\ul`, `\hl` et `\st`), il procède en plaçant le soulignement sous chacune des briques élémentaires. Dans certaines conditions, les logiciels qui font l'impression des documents s'autorisent à déplacer certains caractères de quelques pixels par rapport à leur position normale (le paramètre « maxdrift » de `dvips`, par exemple). Si ce petit décalage n'était pas prévu, lors d'une impression à une résolution faible (en dessous de 300 dpi), les lettres auraient l'air de ne pas être tout à fait à leur place (la largeur des lettres, dans ce type de résolution, n'est pas un nombre entier de pixels). Du coup, en pareil cas, le léger décalage des caractères (ou des traits de souligné, qui sont logés à la même enseigne) peut se traduire par de petits blancs à peine perceptibles dans le trait de souligné (nettement plus perceptibles pour du surligné). Pour empêcher ça, soit on dit au logiciel en question de ne pas se permettre de telles approximations, mais c'est plutôt une mauvaise idée, comme je viens de le dire, soit il faut que le package `soul` fasse des traits un tout petit peu plus longs à droite et à gauche, pour qu'il y ait un recouvrement, et que du coup les traits soient continus malgré le décalage. La commande `\setuloverlap` permet d'indiquer le petit supplément de longueur à ajouter. Par défaut c'est un quart de point (à peut près 7 centièmes de millimètre, soit 70 micromètres), ce qui est bien suffisant.

Le package prévoit explicitement tout un tas de constructions pour leur appliquer le bon traitement. Voyons cette liste.

2.45.2.1 Les accents sont gérés comme il faut, y compris le raccourcis `"` qui est utilisé par `babel` pour l'allemand. Tu peux indiquer de nouvelles commandes pour les accents avec la commande `\soulaccent`.

[50]Dans la pratique, n'importe quel bon macro codeur TeX avec beaucoup d'expérience et de patience saura le dire, mais il n'en sera pas plus avancé pour autant.

`\soulaccent{⟨accent⟩}`

Par exemple, pour indiquer que `\'` est un accent (c'est déjà fait par le package) il suffit d'appeler :

```
\soulaccent{\'}
```

2.45.2.2 Les guillemets anglais et allemands sont prévus par le package, ainsi, s'il voit des séquences qu'il juge être des guillemets, il les laisse ensemble.

Par exemple :

```
\so{``anglais''} et
\so{,,allemand``}
```

produiront : “ a n g l a i s ” et „ a l l e m a n d “ (les guillemets sont conservés, ce qui n'était pas évident). Par contre les guillemets français, ou les ligatures pour les points d'interrogation et d'exclamation espagnols ne sont pas reconnus. Pour t'en convaincre, voilà un petit exemple :

```
\foreignlanguage{spanish}{\so{{!`}espa\~nol!}},
\so{\mbox{\og}fran\c{c}ais\mbox{\fg}}, et
\so{{<<}~fran\c{c}ais~{>>}}.
```

produira : ¡ e s p a ñ o l !, « f r a n ç a i s », et « f r a n ç a i s ».

2.45.2.3 Les maths en ligne, saisies avec la syntaxe de TEX (avec des dollars) passent, pas celles saisies avec la syntaxe LATEX (avec `\(` et `\)`). Par exemple :

```
\so{Maths: $x^3$ vaut $x\times x^2$.}
```

produit : M a t h s : x^3 v a u t $x \times x^2$.

2.45.2.4 Les traits d'unions et tirets sont reconnus, que ce soit `-`, `--` ou `---`. Par exemple :

```
\so{Ce gar\c{c}on est fou --- comme moi.}
```

produit : C e g a r ç o n e s t f o u — c o m m e m o i .

2.45.2.5 Les changements de ligne avec la commande `\\` fonctionnent, mais la variante avec une étoile, de même que l'argument optionnel, ne sont pas reconnus. De même les changements de ligne avec `\linebreak` sont reconnus, mais pas l'argument optionnel.

2.45.2.6 L'espace insécable est reconnue.

2.45.2.7 Ne pas appliquer le traitement sur une partie se demande avec `\soulomit`, par exemple :

```
\so{oui \soulomit{ou} non?}
```

produira : o u i ou n o n ?

2.45.2.8 Les changements de fonte classiques sont reconnus. On peut en ajouter de nouveaux avec la commande `\soulregister`.

`\soulregister{`⟨*commande*⟩`}{`⟨*nb. arguments*⟩`}`

Cette commande peut s'appliquer aux changement des fontes comme `\bfseries`, qui ne prend pas d'argument, ou comme `\textbf` qui en prend un, par exemple :

```
\newcommand\textnom[1]{\textsc{#1}}
\newcommand\fontenom{\scshape}
\soulregister{\textnom}{1}
\soulregister{\fontenom}{0}
\so{\c{C}a \textnom{marche} \fontenom bien\normalfont.}
```

produit : Ç a M A R C H E B I E N.

2.45.2.9 Les notes de pied de page passent, mais l'argument optionnel permettant de changer le numéro de la note ne peut pas être utilisé. Par exemple :

```
\ul{Une note\footnote{La note...} dans du soulign\'e.}
```

produit : Une note[51] dans du souligné.

2.45.2.10 Le groupement d'une césure mène à une erreur. Si tu regroupes dans des accolades (pour traiter comme une seule brique) des lettres qui forment un mot dans lequel un point de césure est possible, le package va se plaindre. Dans ce cas là, il faut utiliser `\mbox`.

2.45.2.11 Les césures explicites ne sont pas gérées par le package, il n'est donc pas autorisé de se servir de la commande `\discretionary`. Ce n'est pas très grave, en français on s'en sert rarement (en allemand, par contre, c'est plus utile, dans le cas où ck se césure en k-k, cas qui ne peut pas être détecté automatiquement de manière simple).

2.45.2.12 L'imbrication des commandes du package ne marche pas. De toutes façons, ce serait plutôt une mauvaise idée.

2.45.2.13 Les changements de fonte cachés dans une commande ne marchent pas spontanément, il faut utiliser `\soulregister` pour déclarer la commande en question.

2.45.2.14 Les éléments à interpréter lors de leur traitement ne marchent pas en général. Par exemple, un appel comme :

```
\so{\romannumeral\year}
```

ne marche pas. Tu peux t'en sortir en passant par une macro, mais ça ne permet pas tout :

```
\def\tmp{\romannumeral\year}
\so\tmp
```

produit : m m v.

En règle générale, tout un tas de constructions ne passent pas bien et doivent être protégées.

[51] La note. Remarque que le texte de la note n'est pas souligné.

2.45.3 Définir des commandes

\sodef{⟨*commande*⟩}{⟨*fonte*⟩}{⟨*lettre*⟩}{⟨*espace*⟩}{⟨*externe*⟩}

Les commandes de la famille de `\so` ajoutent du blanc entre les briques élémentaires issues de la décomposition de leur paramètre. Le blanc indiqué par {⟨*lettre*⟩} est inséré entre deux lettres, le blanc indiqué par {⟨*espace*⟩} est inséré à la place des espaces entre les mots, et le blanc indiqué par {⟨*externe*⟩} remplace, s'ils sont présents, les blancs avant et après l'appel à `\so`. Il vaut mieux définir la taille de ces blancs en utilisant les unités `em` et/ou `ex` qui sont proportionnelles à la fonte en cours d'utilisation : ça permet que l'interlettrage soit lié à la taille de fonte utilisée. Les commandes de changement de fonte indiquées dans le paramètre {⟨*fonte*⟩} sont appliquées à l'ensemble.

Ainsi, si tu veux une commande `\idiote` qui ne fasse que mettre de grands blancs entre les mots, c'est assez facile à obtenir (un blanc normal entre deux mots fait à peu près `.33ex`) :

```
\sodef\idiote{}{0ex}{1em plus 5em}{.33em}
```

si on l'utilise, cette commande produit ce résultat vraiment trop espacé.

Tu peux utiliser `\sodef` pour définir tes propres commandes, comme je viens de le faire, ou pour re-définir la commande `\so`. Si tu as modifié la définition de `\so`, la commande `\resetso` te permet de retrouver la définition originale.

\capsdef{⟨*fonte NFSS*⟩}{⟨*fonte*⟩}{⟨*lettre*⟩}{⟨*espace*⟩}{⟨*externe*⟩}

La commande `\caps`, dont j'ai très peu parlé, ne correspond pas à un usage typographique classique en France, mais plus à des habitudes allemandes. En effet, avec les fontes anciennes en allemand (par exemple celles du package `yfonts`, voir à ce sujet le paragraphe 3.3 page 182) on utilisait un peu d'interlettrage pour les titres tout en capitales ou en petites capitales. Cette commande utilise de petites capitales et un peu d'interlettrage, comme ça se faisait parfois autrefois pour les titres.

Tu peux redéfinir des commandes sur le même modèle grâce à `\capsdef`. Plus exactement, tu peux enrichir son moteur de prise de décision. La commande `\capsdef` sert à ajouter des règles pour lui dire comment faire l'espacement de certaines fontes. Le premier paramètre est celui qu'il utilisera pour savoir quel règlage appliquer (j'y reviendrai), le deuxième indique quel changement de fonte il faut appliquer (spontanément, il applique `\scshape`), et les trois paramètres suivants permettent de spécifier l'interlettrage à appliquer (comme dans `\sodef`).

Le premier paramètre indique un nom de fonte complet, au sens NFSS, avec les 5 dimensions séparées par des « / ». Ainsi la fonte actuelle, en Computer Modern, correspond à `T1/cmr/m/n/10`, c'est-à-dire un codage `T1`, la famille Computer Modern Romain (`cmr`) en graisse moyenne (`m`) et en forme normale (`n`), en taille 10. Quand tu appelles `\capsdef`, le package ajoute la définition en tête de la liste des paramètrages. Quand tu appelles la commande `\caps`, elle balaye la liste (en commencant par la tête, donc par le dernier paramètrage que tu as rentré) et s'arrête sur le premier paramètrage qui ne présente pas de conflit.

Pour mémoire, je te rappelle que la famille correspondant au Palatino est `ppl` et celle correspondant au Times Romain est `txr`. Ainsi, le code suivant indique que pour le Palatino en général, on mettra beaucoup de blanc (`1em` entre chaque lettre), que pour le Times en général on met moins de blanc entre chaque lettre (`.5em`), mais énormément entre les mots (`5em`), et que pour le Palatino italique, on ne met pas de blanc entre les lettres, mais un blanc énorme entre les mots. Bien entendu, ce règlage est parfaitement idiot, mais il se voit bien, ce qui te permettra de contrôler par la suite que ça marche bien.

```
\capsdef{/ppl///}{}{1em}{3em}{1em}
```

```
\capsdef{/txr///}{}{.5em}{2em}{5em}
\capsdef{/ppl//it/}{}{0em}{3em}{.33em}
```

Il est très important que la ligne parlant du Palatino italique (`/ppl//it/`) soit bien en dernier. Quand `\caps` fera sa recherche en commençant par là, si LATEX a du Palatino italique (que ce soit gras ou pas, et quelle que soit la taille), il prendra ce paramètrage, et sinon il continuera sa recherche. Du coup, s'il a du Palatino romain, le premier règlage (`/ppl//it/`) ne cadre pas (il n'est pas en italique), le deuxième règlage (`/txr///`) ne cadre pas non plus (il n'est pas en Times), il va donc se rabattre sur le troisième règlage (`/ppl///`). L'ordre des déclarations est donc très important. En règle générale, il faut commencer par les déclarations les plus générales, et finir par les plus spécifiques. La première règle, celle qui est configurée par défaut, porte sur `////` (elle accepte toutes les fontes). Si l'espacement par défaut ne te plait pas, il te suffit de mettre un règlage `////` en premier dans ta série de déclarations (donc en dernier dans la liste des traitements, juste avant le règlage par défaut), il sera toujours pris (puisqu'il est valable pour toutes les fontes) et masquera entièrement le règlage par défaut.

Un petit exemple d'utilisation pour se convaincre que ça marche. Le source suivant[52] :

```
\begin{center}
\caps{Computer Modern}\par
\fontfamily{ppl}\selectfont
\caps{Palatino romain}\par
\itshape
\caps{Palatino italique}\par
\fontfamily{txr}\upshape\selectfont
\caps{Times Romain}\par
\itshape
\caps{Times italique}\par
\fontfamily{cmr}\upshape\selectfont
\end{center}
```

produira :

COMPUTER MODERN
P a l a t i n o   r o m a i n
Palatino   italique
T i m e s   R o m a i n
T i m e s   i t a l i q u e

`\capssave{`⟨*nom*⟩`}`

`\capsselect{`⟨*nom*⟩`}`

La commande `\capsreset` permet d'annuler tous les changements que tu as fait à la liste de choix avec `\capsdef`. En soi, ce n'est pas super intéressant. Ça le devient quand on la combine avec les commandes `\capssave` et `\capsselect`. La première sauve la liste de réglages sous un nom donné, et la deuxième permet de revenir à une liste précédemment sauvegardée. Tu peux ainsi te définir trois mode d'espacement : un `normal`, qui est le règlage

[52] Les commandes pour changer de fonte sont ici codées en dur, avec les directives de bas niveau de LATEX. Un cas plus réaliste serait d'avoir des espacement légèrement différents pour ces différentes fontes pour produire les titres d'une classe de document, le changement de fonte se faisant pour tout le document avec un des packages de changement de fonte.

par défaut du package ; un `tresblanc` qui met beaucoup de blanc, et un `peublanc` qui en mette très peu. Ça peu se faire comme ça :

```
\capsreset
\capssave{normal}
\capsreset
\capsdef{////}{}{.1ex}{.5em}{.3em}
\capssave{peublanc}
\capsreset
\capsdef{////}{}{1ex}{1em}{1em}
\capssave{tresblanc}
\capsreset
```

À la fin de ces quelques lignes, on est revenu aux règlages par défaut, et on a créé nos trois modes de fonctionnement. Voilà comment tu peux t'en servir :

```
\begin{center}
\capsselect{normal}
\caps{Un essai en mode normal}\\
\capsselect{peublanc}
\caps{Un essai en mode peu blanc}\\
\capsselect{tresblanc}
\caps{Un essai en mode tr\`es blanc}\\
\end{center}
```

produira le résultat suivant[53] :

UN ESSAI EN MODE NORMAL
Un essai en mode peu blanc
U n e s s a i e n m o d e t r è s b l a n c

2.45.4 Utilisation de l'interlettrage

Un des usages classiques de l'interlettrage (probablement le moins condamnable, même si ce n'est pas l'avis de tout le monde) est d'espacer les lettres d'un mot qui sans ça se retrouverait seul sur sa ligne, et ne pourrait donc pas être justifié. C'est le cas, en général, quand on a des colonnes très étroites, comme dans les journaux.

La solution est, en pareil cas, de définir une variété de `\so` qui ajoute un blanc *de taille nulle* entre les lettres, et autorise ce blanc à grandir un peu en cas de besoin. Cette commande est fournie par le package sous le nom `\sloppyword` : tu peux décider de l'appliquer soit à un mot de ton texte qui se retrouverait seul, soit à tout le texte. Si tu l'appliques à tout le texte, cependant, LaTeX fera de l'interlettrage sur *toutes* les lignes où il aura besoin d'ajouter du blanc pour faire sa justification (ce qui est plutôt une mauvaise idée). Il vaut donc mieux utiliser cette commande à la toute fin de la préparation de ton document, une fois que tu es certain que le texte ne changera plus, et que les mots concernés ne changeront plus. Comme tu utilises de l'interlettrage, les règlages d'approches entre les caractères (le fait que certaines combinaisons de lettres doivent être légèrement approchées ou éloignées pour que le résultat visuel soit bon), ainsi que les ligatures prévues par la fonte, ne sont plus possibles.

Si tu dois utiliser de l'interlettrage, il vaut donc *vraiment mieux* le faire sur quelques mots seulement, et le faire en tout dernier dans la préparation de ton document.

[53] À noter que dans les paramètrages que j'ai défini, je n'ai pas fait appel à la commande `\scshape` dans le deuxième paramètre de `\capsdef`, c'est pour ça que le règlage par défaut va passer en petites capitales, mais pas nos deux règlages.

Ce texte est	Ce texte est	Ce texte est
un peu long,	un peu long,	un peu long,
il fait plu-	il fait plu-	il fait plu-
sieurs lignes,	sieurs lignes,	sieurs lignes,
avec certains	avec certains	avec certains
mots un peu	mots un peu	mots un peu
longs qui	longs qui	longs qui
pourraient	pourraient	pourraient fi-
finalement se	finalement se	nalement se
retrouver à	retrouver à	retrouver à
un par ligne.	un par ligne.	un par ligne.

Le texte de la colonne de gauche présente le résultat produit spontanément par LaTeX sur ce texte. Le mot « pourraient » se retrouve seul sur sa ligne, ce qui pose problème.

La colonne du milieu présente le résultat en appliquant la commande `\sloppyword` uniquement au mot problèmatique et à la ligne « longs qui ». Pour tout le reste du texte, LaTeX fait son boulot habituel : les règlages d'approches et les ligatures sont justes.

La colonne de droite présente le même texte où la commande `\sloppyword` a été appliquée à l'ensemble du texte. Toutes les ligatures et les règlages d'approches ont disparu, ce qui produit un résultat douteux. À noter par exemple que la ligature de « finalement » à disparu, et que quasiment toutes les lignes utilisent de l'interlettrage. Le niveau de gris moyen est plus uniforme, mais le texte devient plus dur à lire.

FIG. 2.5: Exemple d'utilisation d'interlettrage en justification étroite avec le package `soul`

La figure 2.5 présente un exemple de texte où un mot se retrouve seul sur sa ligne, dans l'exemple du milieu, seules deux lignes utilisent `\sloppyword`, et pour voir la différence, l'exemple de droite applique `\sloppyword` à l'ensemble du texte. *C'est une chose à ne pas faire.*

2.46 Noms de fichiers : le package `url`

Le package `url` tel que l'a écrit Donald ARSENEAU [8], permet de résoudre ce délicat problème : comment écrire un nom de répertoire, une adresse WEB, une adresse mail, etc., de la manière la moins disgracieuse possible ?

Bin oui, c'est un vrai casse tête. Un truc comme `/usr/lib/texmf/tex/latex2e/contribs/url/url.sty`, c'est extrêmement long, y'a pas d'espace où aller à la ligne, et c'est pas un mot, donc on peut pas faire de césure. De toutes façons, la fonte `\texttt` est programmée pour interdire la césure en partant du principe que cette fonte est utilisée pour des choses à ne pas césurer (commandes TeX, noms de fichiers, de fonctions, de variables. . .).

On peut utiliser ce package à deux niveaux de complexité. Tu n'as normalement besoin que du plus simple des deux.

Le plus simple pour comprendre, pour le moment, c'est un exemple, comme les deux sites CTAN européens :

```
\url{ftp://ftp.dante.de/tex-archive} et
\url+ftp://ftp.tex.ac.uk/tex-archive+
```

qui donnent `ftp://ftp.dante.de/tex-archive` et `ftp://ftp.tex.ac.uk/tex-archive`.

Note dans la seconde forme que tu peux utiliser n'importe quoi à la place du `+`, sauf une accolade ouvrante. Dans ce cas là, tu pourras utiliser librement des accolades dans l'argument d'`\url`.

`\urldef{`⟨*nom-de-commande*⟩`}\url`⟨*texte-de-l'url*⟩

Dans certains cas, si tu utilises `\url` dans l'argument d'une autre macro, par exemple `\footnote`, tu auras la désagréable surprise de constater que ça ne marche pas. C'est pourquoi il est prévu de définir des raccourcis qui *eux* se comporteront bien dans tous les cas. Par exemple :

```
\urldef{\monadresse}\url{bayartb@edgard.fdn.fr}
```

ou encore

```
\urldef{\uneadresse}\url|bayartb%eleves%smig%pc@esiee.fr|
```

Après ces deux appels, les deux commandes `\monadresse` et `\uneadresse` sont définies et valables dans tous les contextes.

`\urlstyle{`⟨*style*⟩`}`

La commande `\urlstyle` permet de changer le style des URLs (en gros, la fonte utilisée). Il existe 4 styles prédéfinis : `tt` qui travail en télétype, `sf` en sans sérifications, `rm` en roman normal, et `same` qui utilise la même fonte que celle en vigueur avant l'appel à `\url`. Si tu lis le source du package, tu découvriras avec stupeur qu'`\url` est défini comme

```
\newcommand{\url}{\begingroup\Url}
```

L'absence d'`\urlstyle` indique que le style retenu est le dernier à avoir été activé dans le même niveau de groupe (comme les changements de taille du type de `\Large`, par exemple).

Comme tu peux vouloir utiliser `\url` pour plusieurs choses (disons des répertoires, des URLs[54] et des adresses mail) il peut être agréable de réserver un style particulier à chaque. Par exemple les adresses mail en Sans Serif, les chemins en télétype, et les URLs aussi en télétype. Pour ce faire :

```
\newcommand\adressemail{\begingroup\urlstyle{sf}\Url}
\newcommand\chemin{\begingroup\urlstyle{tt}\Url}
\newcommand\uneurl{\begingroup\urlstyle{tt}\Url}
```

Normalement, avec ce que je viens de te décrire, tu devrais t'en sortir tranquillement. Le niveau plus élevé dans l'utilisation du package `url` consiste à définir de nouveaux styles ou à changer le comportement de certains caractères, par exemple pour changer les caractères après lesquels on a le droit de changer de ligne. Je ne tiens pas à paraphraser la doc de Donald Arseneau à ce sujet.

[54] Il me vient à l'esprit que tu ne sais peut-être pas ce que c'est qu'une URL. C'est une « Universal Reference Location » ou position de référence universelle, bref, une adresse électronique non-ambigüe qui stipule même dans quelle langue il faut parler à la machine à l'autre bout. Par exemple `ftp://machine.domaine.fr` indique que l'on souhaite parler à `machine.domaine.fr` dans le langage `ftp`.

2.47 Lettrines à la française : le package `lettrine`

Ce package, que nous devons à Daniel Flipo [66], permet de faire de jolies lettrines de début de paragraphe, comme par exemple au début de celui-ci. Il permet de faire un grand nombre de réglages : nombre de lignes à utiliser pour la lettrine, forme de l'espace pris sur le texte (par exemple pour suivre la forme de certaines lettres), etc.

`\lettrine[⟨options⟩]{⟨lettre⟩}{⟨texte⟩}`

Ce package fournit essentiellement une commande : `\lettrine`, qui prend trois paramètres : le premier, optionnel, permet de règler des options sur la lettrine qu'on veut, le deuxième indique la lettrine elle-même, et le troisième le texte à mettre en petites capitales.

Un exemple d'utilisation simple, celui du paragraphe précédent, est :

```
\lettrine[lhang=0.1]{C}{e package,} que nous devons...
```

Les options se passent entre crochets, par exemple sous la forme suivante :

```
\lettrine[lines=3, loversize=0.1]{E}{xemple}
```

Pour chaque option, on peut la changer pour une lettrine, en la précisant entre crochets, ou la changer de manière plus globale, en redéfinissant sa valeur par défaut. La façon de redéfinir cette valeur par défaut changera d'une option à l'autre.

L'option la plus simple à comprendre, et la plus immédiatement utile, est lines qui indique sur combien de lignes se placera la lettrine. Par défaut, la valeur est de 2. Pour redéfinir la valeur par défaut[55] :

```
\setcounter{DefaultLines}{3}
```

L'option lhang permet d'indiquer à quel point la lettrine doit dépasser dans la marge de gauche. La valeur est entre 0 et 1. Dans l'exemple de ce paragraphe la valeur 1, indiquant que la lettre doit dépasser totalement dans la marge, a été choisie[56]. La valeur par défaut est 0. Une valeur de 0,1 est classique, par exemple pour les lettres arrondies sur le côté gauche (comme le C du premier exemple). Pour changer la valeur par défaut :

```
\renewcommand{\DefaultLhang}{1}
```

L'option loversize permet d'indiquer que la lettrine doit être plus grande que le nombre de lignes réservées dans le texte, pour qu'elle dépasse au dessus de la première ligne. La valeur doit être comprise entre -1 (exclus) indiquant de réduire la taille d'une ligne, et 1 indiquant d'augmenter la taille d'une ligne. La valeur par défaut est 0. Ce paragraphe est composé avec une valeur de 1. Pour changer la valeur par défaut :

```
\renewcommand{\DefaultLoversize}{0.5}
```

Quand on souhaite réhausser la lettrine sans pour autant que sa taille soit changée, par exemple pour la lettre Q initiale de ce paragraphe, on peut utiliser l'option lraise. Quand la valeur est positive, la lettre est déplacée vers le haut ; quand elle est négative c'est vers le bas. La valeur par défaut est 0. La lettrine de ce pragraphe a été réhaussée de 0,2 : en effet

[55] Pour plus d'informations sur le fonctionnement, et le sens précis, de la commande `\setcounter`, reporte-toi au chapitre neuvième page 571 qui traite de la programmation.

[56] Dans la version en deux colonnes de ce manuel, pour peu que la lettrine arrive dans la colonne de droite, le résultat est très laid : la lettrine apparaît en sur-impression entre les deux colonnes, recouvrant même une partie de la colonne de gauche.

le bas de la lettre Q descend légèrement sous la ligne, et aurait donc pu toucher une lettre montante qui se serait trouvée en dessous. Pour changer la valeur par défaut :

```
\renewcommand{\DefaultLraise}{0.2}
```

Voici les options permettant de règler l'espacement autour de la lettrine. La première est `findent` (pour *first indent*), elle indique de quelle taille doit être le blanc séparant la lettrine du bloc de texte en petite capitales. Par défaut, c'est `0pt`. Cette indentation change pour la deuxième ligne (et les suivantes, s'il y a lieu), on utilise alors la valeur de l'option `nindent` (pour *next indent*). Par défaut c'est `0.5em` c'est-à-dire la moitié de la largeur d'un M. Dans les règlages par défaut, la première ligne est moins indentée que la deuxième pour permettre une bonne lecture : le premier mot se retrouve ainsi lisible, et le premier mot des lignes suivantes est bien détaché de la lettrine. Enfin, si tu souhaites que le texte suive la forme d'une lettre, comme dans ce paragraphe, tu peux utiliser l'option `slope`. C'est un longueur qui sera ajoutée à l'indentation des lignes à partir de la troisième, et ce à chaque ligne. Pour composer ce paragraphe, j'ai utilisé les options suivantes :

```
\lettrine[lines=6, slope=-5pt]{V}{oici} les ...
```

Alors que pour ce paragraphe là, il a fallut utiliser des valeurs très différentes. D'abord, réduire `findent`, pour que la première lettre de « Alors » se rapproche du sommet du A. Ensuite, utiliser une valeur de `slope` différente pour que le texte suive la lettre en *s'éloignant* du bord de la page. Enfin, il a fallu changer la valeur de `nindent` pour que toutes les lignes sauf le première soient bien détachées de la lettre initiale. La valeur choisie pour `findent` est de `-25pt`, ce qui rapproche le texte de la première ligne. Pour `nindent`, j'ai choisi `15pt`, c'est-à-dire que la deuxième ligne commence 15 points plus à droite que la première, ce qui permet de détacher les lignes suivantes. Enfin, j'ai indiqué une valeur pour `slope` de `5pt`, pour que le texte suive la pente de la lettre.

Enfin, un point est à retenir : spontanément, de base, LaTeX n'accepte pas toutes les tailles de fontes. Il a été prévu pour des systèmes où le disque coûte cher, et pour travailler avec des fontes en taille fixe (du temps où les fontes vectorielles coûtaient trop de puissance à afficher). Du coup, il a une liste des tailles disponibles, et quand on lui demande une taille précise, il prend celle qui est la plus proche dans sa liste. La taille maximale disponible en standard dans LaTeX est de 35 points (approx.), ce qui correspond à peu près à ce dont tu as besoin pour une lettrine sur 3 lignes. Pour obtenir plus, il faudra ruser :

- utiliser des fontes pour lesquelles toutes les tailles sont disponibles, par exemple des fontes PostScript Type 1 ; ou
- utiliser un des packages qui déclarent des tailles suppérieures ; ou
- redéclarer la fonte dont tu as besoin, en disant à LaTeX d'utiliser des tailles plus grandes, et utiliser cette fonte là pour faire la lettrine.

Pour ma part, comme je ne souhaitais pas écrire mes exemples en Times, et que je voulais vraiment des grandes tailles (plus que ce que prévoit `moresize`), j'ai opté pour la troisième solution.

Une ligne magique, pour déclarer la fonte, et une ligne pour dire à la commande `\lettrine` d'utiliser ma fonte. Pour ce qui est de la ligne magique, je redéclare la fonte normale de mon livre sous un autre nom (`bba`), et j'indique d'utiliser toutes les tailles possibles pour cette fonte. En deux lignes, ça donne ça[57] :

[57] Pour plus de détails sur l'explication de cette manipulation, voir au chapitre neuvième page 571. Les commandes `\DeclareFontShape`, `\fontshape` et `\selectfont` sont expliquées pour leur part au paragraphe 11 page 607.

```
\DeclareFontShape{T1}{cmr}{m}{bba}{<->genb*ecrm}{}
\renewcommand\LettrineFontHook{\fontshape{bba}\selectfont}
```

2.48 Lettrines avant : le package `dropcaps`

CE package, que nous devons à la générosité de Fred J. LAUWERS [119] permet de faire ce qu'en français il est convenu d'appeler des lettrines. C'est un package ancien, très ancien. Il date d'avant LaTeX 2$_\varepsilon$ (avant 1994 pour fixer les idées). Moins puissant que le package `lettrine`, mais aussi moins gourmand, si tu as besoin de l'utiliser sur une très vieille machine par exemple.

Il fonctionne de manière relativement simple, puisqu'il définit une seule commande qui sera ensuite utilisée pour créer cette lettrine. Ainsi pour la lettrine de ce premier paragraphe, j'ai fait appel à la commande :

```
\bigdrop{-10pt}{3}{eccc1000}{Ce} package, que ...
```

`\bigdrop{`⟨*longueur*⟩`}{`⟨*lignes*⟩`}{`⟨*fonte*⟩`}{`⟨*texte*⟩`}`

Relevons la signification des arguments dans l'ordre :

-10pt indique que la « lettrine » doit dépasser de 10 points dans la marge, lorsque l'on a des lettres italiques, ou comme le C un peu arondi, c'est agréable.

3 indique que la lettrine doit s'étendre sur trois lignes.

eccc1000 est la fonte que j'ai choisie d'utiliser, j'en conviens, c'est pas tout le monde qui connaît les noms des fontes en dur[58], mais j'y peux rien.

Ce est le texte de la lettrine.

Le seul aspect franchement désagréable de ce package est qu'il pose une question durant la compilation pour savoir si on utilise ou pas le programme `dvips` pour l'impression. C'est en effet assez désagréable : ça interrompt le travail. Je pense que le jour où ce package sera porté vers LaTeX 2$_\varepsilon$, le choix se fera par option de chargement.

2.49 Lettrines à peine mieux : le package `dropping`

Le package `\dropping`, que nous devons à Mats DAHLGREN, reprend le concept (et l'implémentation) du package `dropcaps` en automatisant le choix de la fonte. Il fournit les mêmes commandes que `dropcaps` et ajoute la commande `\dropping` un peu plus simple à utiliser.

Pour indiquer que tu veux le mode de fonctionnement associé à `dvips`, ce qui est de nos jours la norme pour l'immense majorité des logiciels autour de TeX, il faut lui indiquer l'option dvips. En fait, il connaît les mêmes options que le package `graphics`, mais ce n'est pas très important : il divise le monde en deux, avec ou sans `dvips`. Par défaut, d'ailleurs, il prend l'option dvips. Si tu tombes sur un logiciel ancien, qui n'est pas compatible avec ce mode de fonctionnement (*i.e.* qui ne soit pas capable de générer des fontes à la volée), tu peux essayer l'option other qui se limite aux tailles connues et pré-calculées, mais ce n'est a mon avis pas utile.

[58]Pour info, voilà comment tu dois lire ce nom de fonte : il s'agit des fontes EC, c'est-à-dire l'aspect des Computer Modern de KNUTH, à savoir les fontes que tu vois le plus couramment dans ce livre, en petites capitales (cc) dans un corps de 10,00 points. En fait, la seule partie vraiment délicate est donc le cc qui ne peut pas se deviner tout seul.

`\dropping[⟨longueur⟩]{⟨lignes⟩}{⟨texte⟩}`

LA commande `\dropping` est basée sur la commande `\bigdrop` de `dropcaps`, sauf qu'elle calculera toute seule la fonte qu'il faut choisir en fonction de ce que tu indiques dans le paramètre `{`⟨*texte*⟩`}`. Comme pour les autres packages qui font des lettrines, il ne faut pas que le paragraphe concerné se retrouve en bas de page : le changement de page donnerait un résultat surprenant (par exemple la lettrine qui dépasse sur le pied de page, et les lignes suivantes, sur la page d'après, qui seraient indentées).

À noter que l'on n'a pas le droit de mettre ce qu'on veut comme changement de fonte dans le paramètre `{`⟨*texte*⟩`}`. Ce ne doivent être que des changements de fonte directs (comme `\scshape` ou `\bfseries`, pas de `\textsc` ou de `\textbf`). De même juste après le changement de fonte il doit y avoir un espace, qui ne sera pas pris en compte. Ainsi le début de mon paragraphe ressemble à ça :

```
\dropping{3}{\scshape{} La}commande...
```

Si tu oublies les accolades et l'espace, le package ne marchera pas, et produira une erreur peu compréhensible, quelque chose comme :

```
Runaway argument?
\fi \protect \bigdrop {0pt}{3}{\xprt@rgtr }{\xprt@rgfr }commande \TidxCde \ETC.
! Paragraph ended before \@split was complete.
<to be read again>
                   \par
l.20

?
```

Bien entendu, tout cela reste quand même beaucoup moins puissant que ce que propose le très beau package `lettrine`. En particulier, pour obtenir de grandes lettres, le package prend une petite fonte (de la taille du texte courant) et l'agrandit pour l'amener à la bonne hauteur. Le package `lettrine` fait un travail nettement plus intéressant sur ce point-là.

2.50 Une fonte perdue : le package `ssqquote`

Ce petit package, écrit par Ulrik VIETH [213] permet de faire des citations en fin de chapitre comme fait KNUTH dans le TEXbook et je trouve que c'est assez joli. Il fait appel à une fonte par ailleurs inutilisée de la famille des Computer Modern[59]

[59] Ce package est ancien, il n'a pas évolué depuis mai 1994. Certains aspects ne marchent plus : la fonte qu'il utilise existe en codage OT1 et en codage T1. Mais, en 1994, les fontes T1 n'étaient pas encore finalisées, et portaient un nom provisoire : les fontes `dc`, qui sont devenues par la suite les fontes `ec`. Ce package cherche donc à utiliser les fontes de 1994, et en particulier, il cherche les fontes `dcssq8` et `dcssqi8`. Ces fontes n'existent plus de nos jours.

Tu as donc deux possibilités.

1. Tu utilises la variété OT1, qui est diffusée avec toutes les versions connues de TEX depuis plus de 20 ans. Pour ce faire, tu appelles
   `\renewcommand\encodingdefault{OT1}`
   avant l'environnement `chapterquotes` et
   `\renewcommand\encodingdefault{T1}`
   après.
2. Tu utilises la version T1 des fontes `ec`. Pour ce faire, tu récupère le fichier `t1cmssq.fd` à l'URL :
   `http://jmpl.fr.eu.org/JMPL/t1cmssq.fd`
   et tu le mets à la place du fichier `T1cmssq.fd` qui est diffusé avec le package (oui, l'ancien fichier avait une majuscule, et le nouveau n'en a pas : pour éviter les problèmes avec certains systèmes

Ce package propose une nouvelle famille de fonte, accessible via la commande `\textssq` (ou `\ssqfamily`). La seule variante disponible est le penché (l'italique renvoit vers le penché). Il n'y a pas de gras ou de petites capitales[60].

La fonte en question est assez voisine du sans-serif classique de Computer Modern. Pour comparaison :

`\textsf`	`\textssq`
Texte d'exemple	Texte d'exemple
Texte d'exemple	*Texte d'exemple*

Ce package fournit également un environnement, `chapterquote`, qui fait des citations de fin de chapitre comme on en trouve dans les livres de KNUTH (voir, par exemple, le TEXbook [108]). Dans cet environnement, chaque citation doit commencer par `\nextquote`, y compris la première. La macro `\author` permet de mettre en forme le nom de l'auteur, et `\title` le titre, s'il y a lieu. Un petit détail : la macro `\author`, dans l'environnement `chapterquote`, attends le nom de l'auteur *sans accolades*, ou du moins, sans en avoir besoin, puis quelque chose (date, remarque, édition, etc.) entre parenthèses. Ainsi, dans l'exemple ci-après, j'ai daté la première citation entre parenthèses, et j'ai donné le nom du personnage auquel est attribué la réplique, pour le seconde citation.

```
\begin{chapterquotes}
\nextquote
Y fait B\^o\^o\^o\^o\^o\^o-\^o\^o\^o\^o-\^o\^o\^o\^o
\author P. Vincent (Un jour de printemps)
\nextquote
Nous allons \‘a jamais vers demain.
\author J.R.R. Tolkien,
\title{The Lord of the Rings}
(Bilbo Baggins)
\end{chapterquotes}
```

Y fait Bôôôôôô-ôôôô-ôôôô

— P. Vincent (Un jour de printemps)

Nous allons à jamais vers demain.

— J.R.R. Tolkien,
The Lord of the Rings
(Bilbo Baggins)

d'exploitation qui ne font pas la différence entre majuscules et minuscules, LATEX n'utilise plus que des minuscules depuis quelques années, mais le package est plus ancien que ça).

Si la notion de codage ne t'évoque rien, reporte-toi au chapitre neuvième, paragraphe 11 page 607 qui traite de NFSS.

[60]Le package utilise des normes de développements de 1994. Il fonctionne bien dans la majorité des cas, mais est succeptible de produire des messages d'erreur étranges dans certains cas (utilisation de cette fonte dans des maths, par exemple), ou de ne pas fonctionner dans certains cas particulièrement tordus (appel d'une note de pied de page dans une note de fin de volume, par exemple). J'ai écrit un petit package de quelques lignes, `myssqquote`, qui redéclare les commandes `\textssq` et `\ssqfamily` selon des méthodes plus modernes.

Ce package est disponible à l'URL : `http://jmpl.fr.eu.org/JMPL/myssqquote.sty`