TEX pour l'Impatient

Ce livre, *TEX pour l'Impatient*, contient le tutorial et l'information de référence sur tous les dispositifs de plain TEX et des primitives de TEX.

Pour Jodi.
—P.W.A.

à la mémoire de mon père,
qui a eu foi en moi.
—K.A.H.

Pour Dan.
—K.B.

Préface

TEX de Donald Knuth, un système de composition automatisé, fournit presque tout ce qui est requis pour la composition de haute qualité, aussi bien des mathématiques que du texte ordinaire. Il est particulièrement reconnu pour sa flexibilité, ses césures superbes et sa capacité de choisir des coupures de ligne esthétiquement satisfaisantes. En raison de ses possibilités extraordinaires, TEX est devenu le système de composition principal pour les mathématiques, les sciences et techniques et a été adopté comme norme par la société mathématique américaine. Un programme compagnon, METAFONT, peut construire les lettres arbitraires comprenant, en particulier, tous les symboles qui pourraient être nécessaires dans les mathématiques. TEX et METAFONT sont largement disponibles au sein de la communauté scientifique et technologique et ont été mis en application sur une variété d'ordinateurs. TEX n'est pas parfait—il lui manque un support intégré des graphiques et certains effets tels que les barres de révision sont très difficiles à produire—mais ces inconvénients sont loin d'être supérieurs à ses avantages.

TEX pour l'Impatient est prévu pour servir aux scientifiques, mathématiciens, et typographes pour lesquels TEX est un outil utile plutôt que d'un intérêt primaire, aussi bien que pour les informaticiens qui ont un vif intérêt pour TEX dans son propre intérêt. Nous avons l'intention également de servir aussi bien les nouveaux venus que ceux qui sont déjà familiarisé avec TEX. Nous supposons que nos lecteurs sont habitués au fonctionnement des ordinateurs et qu'ils veulent obtenir l'information dont ils ont besoin aussi rapidement que possible. Notre but est de fournir ces informations clairement, avec concision, et de manière accessible.

Ce livre fournit donc un éclairage lumineux, un guide vaillant et donne des cartes détaillées pour explorer et utiliser TEX. Il vous permettra de maîtriser TEX rapidement par l'enquête et l'expérience, mais il ne vous mènera pas par la main dans le système TEX en entier. Notre approche est de vous fournir un manuel de TEX qui vous facilite la recherche des quelques informations dont vous avez besoin. Nous expliquons tout le répertoire des commandes de TEX et des concepts dont elles dépendent. Vous ne perdrez pas de temps à explorer des sujets que vous ne voulez pas voir ni besoin.

Dans les premières sections, nous vous fournissons également assez d'indication pour que vous puissiez commencer si vous n'avez jamais utilisé TEX avant. Nous supposons que vous avez accès à un exécutable TEX et que vous savez utiliser un éditeur de texte, mais nous ne supposons pas beaucoup plus au sujet de vos connaissances. Puisque ce livre est organisé pour la référence, vous continuerez toujours à le trouver utile quand vous vous familiariserez avec TEX. Si vous préférez commencer par une excursion soigneusement guidée, nous vous recommandons d'avoir lu une première fois *The TEXbook* de Knuth (voir la page 18 pour une citation), en passant au-dessus des "sections dangereuses", puis de revenir à ce livre pour de l'information additionnelle et des références pendant que vous commencez à utiliser TEX. (les sections dangereuses recouvrent des sujets avancés de *The TEXbook*).

La structure de TEX est vraiment très simple : un document TEX se compose du texte ordinaire entremêlé avec des commandes qui donnent à TEX des instructions complémentaires sur la façon de composer votre document. Les choses comme des formules de maths contiennent beaucoup de commandes, alors que le texte d'explication en contient relativement peu.

La partie fastidieuse de l'étude de TEX est d'apprendre les commandes et les concepts sous-entendus dans leurs descriptions. Ainsi nous avons consacré la majeure partie du livre à définir et à expliquer les commandes et les concepts. Nous avons également fourni des exemples en montrant le résultat composé avec TEX et la source correspondante, des conseils pour résoudre des problèmes communs, des informations sur les messages d'erreur, et ainsi de suite. Nous avons fourni des références croisées par numéro de page et un index complet.

Nous avons classé les descriptions de commandes pour que vous puissiez les rechercher par fonction ou alphabétiquement. Le classement fonctionnel est ce dont vous avez besoin quand vous savez ce que vous voulez faire mais que vous ne savez pas quelle commande peut le faire pour vous. Le classement alphabétique est ce dont vous avez besoin quand vous connaissez le nom d'une commande mais pas exactement ce qu'elle fait.

Nous devons vous avertir que nous n'avons pas essayé de fournir une définition complète de TEX. Pour cela vous aurez besoin de *The TEXbook*, qui est la source originale d'information sur TEX. *The TEXbook* contient également beaucoup d'informations sur les détails d'utilisation de TEX, en particulier sur la composition des formules mathématiques. Nous vous le recommandons fortement.

En 1989, Knuth a fait une révision importante de TEX afin de l'adapter aux jeux de caractères 8 bits requis pour effectuer des compositions dans des langues autres que l'anglais. La description de TEX dans ce livre tient compte de cette révision (voir le p. 18).

Vous pouvez utiliser une forme spécialisée de TEX comme LATEX ou $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-TEX (voir les p. 18). Bien que ces formes spécialisées soient d'un seul bloc, vous pouvez vouloir juste utiliser certaines des facilités de TEX

lui-même, afin d'employer la commande la plus fine que seul TEX peut fournir. Ce livre peut vous aider à apprendre ce que vous devez connaître à propos de ces facilités sans devoir vous renseigner sur beaucoup d'autres choses qui ne vous intéressent pas.

Deux d'entre nous (K.A.H. et K.B.) ont été généreusement soutenus par l'université du Massachusetts à Boston pendant la préparation de ce livre. En particulier, Rick Martin a permis aux machines de fonctionner et Robert A. Morris et Betty O'Neil les ont rendus disponibles. Paul English d'Interleaf nous a aidés à produire les épreuves d'un dessin de couverture.

Nous souhaitons remercier les relecteurs de notre livre : Richard Furuta de l'université du Maryland, John Gourlay d'Arbortext, Inc., Jill Carter Knuth et Richard Rubinstein de la Digital Equipment Corporation. Nous avons pris à cœur leurs critiques constructives et sans compter du manuscrit original, et le livre a considérablement bénéficié de leurs perspicacités.

Nous sommes particulièrement reconnaissant à notre rédacteur, Peter Gordon d'Addison-Wesley. Ce livre était vraiment son idée et dans tout son développement il a été une source d'encouragement et conseil valable. Nous remercions son aide chez Addison-Wesley, Helen Goldstein, qui nous a aidés de tant de manières, et de Loren Stevens d'Addison-Wesley de sa compétence et de son énergie dans le cheminement de ce livre dans le processus de production. Sans oublier notre rédactrice de copie, Janice Byer, sans qui un certain nombre de petite mais irritantes erreurs seraient demeurées dans ce livre. Nous apprécions sa sensibilité et goûtions en corrigeant ce qui nous était nécessaire de corriger tout en lui laissant ce que nous n'avions pas eu besoin de corriger. En conclusion, nous souhaitons remercier Jim Byrnes de Prometheus Inc. d'avoir rendu cette collaboration possible en nous présentant.

Deerfield, Massachusetts P. W. A.
Manomet, Massachusetts K. A. H., K. B.

Préface à l'édition libre : Ce livre a été à l'origine édité en 1990 par Addison-Wesley. En 2003, il a déclaré épuisé et Addison-Wesley a généreusement rendu tous les droits à nous, les auteurs. Nous avons décidé de rendre le livre disponible sous forme de source, sous la licence de documentation libre de GNU, ceci étant notre manière de soutenir la communauté qui a soutenu le livre en premier lieu. Voyez la page de copyright pour plus d'information sur l'autorisation.

Les illustrations qui faisaient partie du livre original ne sont pas incluses ici. Certaines des polices ont également été changées ; maintenant, seules des polices librement disponibles sont utilisées. Nous avons laissé les hirondelles et l'information de galée sur les pages, pour identification. Une vieille version d'Eplain a été employée pour la produire ; voyez le dossier `eplain.tex` pour des détails.

Nous ne projetons pas de faire de changements ou additions au livre nous-mêmes, excepté la correction d'erreurs qui nous seront rapportées et peut-être, l'inclusion des illustrations.

Notre distribution du livre est à `ftp://tug.org/tex/impatient`. Vous pouvez nous atteindre par courriel à `impatient@tug.org`.

Table des matières abrégée

1 \ Utiliser ce livre ■ 1
2 \ Utiliser TEX ■ 7
3 \ Exemples ■ 23
4 \ Concepts ■ 45
5 \ Commandes pour composer des paragraphes ■ 103
6 \ Commandes pour composer des pages ■ 139
7 \ Commandes pour les modes horizontaux et verticaux ■ 159
8 \ Commandes pour composer des formules mathématiques ■ 195
9 \ Commandes pour des opérations générales ■ 229
10 \ Trucs et astuces ■ 273
11 \ Comprendre les messages d'erreur ■ 293
12 \ Un abrégé de macros utiles ■ 301
13 \ Sommaire des commandes ■ 323

GNU Free Documentation License ■ 353
Index ■ 363

Table des matières

1 \ Utiliser ce livre ■ 1

Conventions syntaxiques □ 3

Descriptions des commandes □ 3

2 \ Utiliser TEX ■ 7

Transformer la source en encre □ 7
- les programmes et fichiers dont vous avez besoin · 7
- exécuter TEX · 9

Préparer un fichier source □ 10
- Commandes et séquences de contrôle · 10
- Arguments · 11
- Paramètres · 12
- Espaces · 12
- Commentaires · 13
- Ponctuation · 13
- Caractères spéciaux · 15
- Groupes · 15
- formules mathématiques · 16

Comment marche TEX □ 16

Nouveau TEX contre ancien TEX □ 18

Ressources □ 18

Ressources en français □ 19
- TEX · 20
- LATEX · 20
- METAFONT · 20

3 \ Exemples ■ 23

Saisir du texte simple □ 24

Indentation □ 26

Polices et caractères spéciaux □ 28

Espacement interligne □ 30

Espacement, traits et boîtes □ 32

Odds and ends □ 34

Utiliser des polices venant d'ailleurs □ 36

Un tableau tracé □ 38

Composer des mathématiques □ 40

Plus de mathématiques □ 42

4 \ Concepts ■ 45

5 \ Commandes pour composer des paragraphes ■ 103

Caractères et accents □ 103
- Lettres et ligatures pour alphabets Européens · 103
- Symboles spéciaux · 104
- Caractères Arbitraires · 105
- Accents · 106
- Defeating boundary ligatures · 107

Sélectionner des polices □ 108
- Polices particulières · 108
- Styles de caractère · 109

Majuscule et minuscule □ 109

Espace inter-mot □ 110

Centrer et justifier les lignes □ 115

Formation des paragraphes □ 116
- débuter, finir et indenter des paragraphes · 116
- Formation de paragraphes entiers · 120

coupures de lignes □ 126
- Encourager ou décourager les coupures de ligne · 126
- paramètres de coupure de lignes · 128
- Césure · 132

Entêtes de section, listes et théorèmes □ 135

6 \ Commandes pour composer des pages ■ 139

Espaces inter-ligne et inter-paragraphe □ 139

Coupures de page □ 142
- Encourager ou décourager des coupures de page · 142
- Paramètres de coupure de page · 144

Gabarit de page ▫ 146
Paramètres de description de page · 146
Numéros de page · 148
Lignes d'entête et de pied de page · 149
Marques · 150

Insertions ▫ 151
Pieds de page · 151
Insertions générales · 152

Modifier la routine de sortie ▫ 154

Séparer des listes verticales ▫ 155

7 \ Commandes pour les modes horizontaux et verticaux ■ 159

Produire des espaces ▫ 159
Espaces horizontaux de largeur fixe · 159
Espaces verticaux de largeur fixe · 160
Espace de taille variable · 161

Manipuler des boîtes ▫ 166
Construire des hbox et des vbox · 166
Remplir et récuperer le contenu de boîtes · 170
Déplacer des boîtes · 172
Dimensions des registres de boîtes · 173
Struts, fantomes et boîtes vides · 173
Paramètres faisant partie des boîtes mal formées · 175

Retrouver le dernier élément d'une liste ▫ 177

filets et règlures ▫ 178

Alignements ▫ 181
Alignements tabulés · 181
Alignement généraux · 184

8 \ Commandes pour composer des formules mathématiques ■ 195

Parties simples de formules ▫ 195
Lettres grecques · 195
Divers Symboles mathématiques ordinaires · 196
Opérations binaires · 197
Relations · 198
Délimiteurs gauches et droits · 199
Flèches · 200
Fonctions mathématiques nommées · 201
Grands opérateurs · 202
Ponctuation · 204

Puissances et indices ▫ 205
Choisir et utiliser des modèles · 206

Symboles composés □ 207
Accents mathématiques · 207
Fractions et autres opérations empilées · 208
Points · 211
Délimiteurs · 212
Matrices · 213
Racines et radicaux · 214
Numéros d'équation □ 215
Affichages multi-ligne □ 216
Polices dans des formules mathématiques □ 217
Construire des symboles mathématiques □ 219
Rendre des délimiteurs plus grand · 219
Parties de grands symboles · 220
Aligner des parties d'une formule □ 220
Aligner des accents · 220
Aligner du matériel verticalement · 221
Produire des espaces □ 222
Espaces mathématiques de largeur fixe · 222
Espaces mathématiques de largeur variable · 223
paramètres d'espacement pour les affichages · 224
autres paramètres d'espacement pour les mathématiques · 225
Catégoriser des constructions mathématiques □ 226
Actions spéciales pour des formules mathématiques □ 226

9 \ Commandes pour des opérations générales ■ 229

Nommer et modifier des polices □ 229
Convertir l'information en tokens □ 232
Nombres · 232
Information environmentale · 233
Valeurs des variables · 234
Groupement □ 235
Macros □ 238
Définir des macros · 238
Autre définitions · 240
Contrôler le développement · 241
Tests conditionnels · 243
Actions répétées · 248
Ne rien faire · 249
Registres □ 250
Utiliser des registres · 250
Nommer et réserver des registres, etc. · 252
Faire de l'arithmétique dans des registres · 253

Terminer l'exécution ▫ 255

Entrée et sortie ▫ 255
Opérations sur des fichiers d'entrée · 255
Opérations sur des fichiers de sortie · 257
Interpréter des caractères entrés · 259

Contrôler l'interaction avec TEX ▫ 260

Aide au diagnostique ▫ 261
Afficher des données internes · 261
Spécifier ce qui est tracé · 264
Envoyer des messages · 269

Initialiser TEX ▫ 271

10 \ Trucs et astuces ■ 273

Corriger de mauvaises coupures de page ▫ 273

Préserver la fin d'une page ▫ 275

Garder de l'espace en haut d'une page ▫ 276

Corriger de mauvaise coupure de ligne ▫ 276

Corriger des boîtes trop ou pas assez pleines ▫ 276

Retrouver des espaces entre-mots perdus ▫ 278

éviter des espaces entre-mots non désirés ▫ 278

éviter un excès d'espace autour d'un affichage ▫ 279

éviter un excès d'espace après un paragraphe ▫ 280

Changer la forme du paragraphe ▫ 280

Mettre des paragraphes dans une boîte ▫ 281

dessiner des lignes ▫ 281

Créer des entêtes ou des pieds de page multi-lignes ▫ 282

Trouver des accolades orphelines ▫ 283

Fixer des dimensions ▫ 284

Créer des polices composites ▫ 284

Reproduire du texte verbatim ▫ 285

Utiliser des macros externes ▫ 288

Changer des codes de catégorie ▫ 288

Faire des fichiers de macro plus lisibles ▫ 289

11 \ Comprendre les messages d'erreur ■ 293

12 \ Un abrégé de macros utiles ■ 301

Préliminaires □ 301
Affichages □ 305
heure du jour □ 307
Listes □ 308
Listing verbatim □ 310
Table des matières □ 311
Références croisées □ 312
Environnements □ 314
Justification □ 316
Tables □ 317
Notes de pied de page □ 318
Double colonnes □ 319
Terminer □ 321

13 \ Sommaire des commandes ■ 323

GNU Free Documentation License ■ 353

PREAMBLE □ 353
APPLICABILITY AND DEFINITIONS □ 354
VERBATIM COPYING □ 355
COPYING IN QUANTITY □ 356
MODIFICATIONS □ 356
COMBINING DOCUMENTS □ 358
COLLECTIONS OF DOCUMENTS □ 359
AGGREGATION WITH INDEPENDENT WORKS □ 359
TRANSLATION □ 359
TERMINATION □ 360
FUTURE REVISIONS OF THIS LICENSE □ 360

Index ■ 363

lisez ceci d'abord

Si vous débutez en TEX:

- Lisez en premier les sections 1–2.
- Recherchez dans les exemples de la section 3 les choses qui ressemblent à ce que vous voulez faire. Recherchez toutes les commandes en relations dans le "sommaire des commandes", section 13. Utilisez les références de page pour trouver les descriptions les plus complètes sur ces commandes et d'autres qui leur sont similaires.
- Recherchez les mots inconnus dans "Concepts", section 4, en utilisant la liste à la fin du livre pour trouver l'explication rapidement.
- Expérimentez et explorez.

Si vous connaissez déjà TEXou si vous éditez ou modifiez un document TEX que quelqu'un d'autre a créé:

- Pour un rappel rapide de ce que fait telle ou telle commande, regardez dans la section 13, "résumé des commandes". Il est alphabétique et comprend des références de page pour de plus complètes descriptions des commandes.
- Utilisez les groupements fonctionnels de descriptions de commande pour trouver celles liés à une commande particulière que vous connaissez déjà, ou pour trouver une commande qui atteint un objectif particulier.
- Utilisez la section 4, "concepts", pour avoir l'explication de n'importe quel concept que vous ne comprenez pas, voulez comprendre avec plus de précision ou avez oublié. Utilisez la liste à la fin du livre pour trouver un concept rapidement.

1 Utiliser ce livre

Ce livre est un guide et un manuel de bricolage pour TEX. Dans cette section nous vous présentons comment utiliser ce livre avec le maximum de bénéfice.

Nous vous recommandons de lire ou survoler les sections 1 à 3, qui vous indiquent ce que vous devez savoir afin de pouvoir commencer à utiliser TEX. Si vous avez déjà de l'expérience dans l'utilisation de TEX, il vous sera toujours utile de connaître quels types d'information sont dans ces sections du livre. Les sections 4 – 10, qui occupent la majeure partie du reste du livre, sont conçues pour être atteintes directement. Néanmoins, si vous êtes le genre de personne qui aime lire les manuels de référence, vous constaterez qu'il *est* possible de procéder séquentiellement si vous êtes prêt à prendre beaucoup de détours au début.

Dans la section 2, "Utiliser TEX", nous expliquons comment produire un document TEX à partir d'un fichier source TEX. Nous décrivons également les conventions pour préparer ce fichier source, expliquons un peu la façon dont TEX fonctionne et vous présentons des ressources supplémentaires disponibles. La lecture de cette section vous aidera à comprendre les exemples dans la section suivante.

La section 3, "exemples", contient une suite d'exemples qui illustrent l'utilisation de TEX. Chaque exemple se compose d'une page de sortie ainsi que de la source que nous avons utilisée pour la créer. Ces exemples vous orienteront et vous aideront à localiser le matériel le plus détaillé dont vous aurez besoin pour faire ce que vous voulez. En voyant quelles commandes sont utilisées dans la source, vous saurez où rechercher des informations plus détaillées sur la façon de réaliser les effets montrés dans la sortie. Les exemples peuvent également servir de modèles à des documents simples, bien que nous devions vous avertir que parce que nous avons essayé de compiler une variété de commandes TEX de dans un nombre de pages restreint, les exemples ne sont pas nécessairement de bonnes ou complètes illustrations de la création d'imprimés.

Quand vous lisez l'explication d'une commande, vous pouvez rencontrer quelques termes techniques peu familier. Dans la section 4, "concepts", nous définissons et expliquons ces termes. Nous discutons également d'autres matières dont il n'est pas fait mention ailleurs dans le livre. L'intérieur de la couverture de dos du livre contient une liste de tous les concepts et les pages où elles sont décrites. Nous vous proposons de tirer une copie de cette liste et de la maintenir près de vous pour que vous puissiez identifier et rechercher immédiatement un concept peu familier.

Les commandes de TEX sont son vocabulaire primaire et la plus grande partie de ce livre est consacrée à les expliquer. Dans les sections 5 à 9 nous décrivons les commandes. Vous trouverez des informations générales au sujet des descriptions de commande sur la page 3. Les descriptions de commande sont arrangées fonctionnellement, plutôt comme un thesaurus, ainsi si vous savez ce que vous voulez faire mais ne savez pas quelle commande le fait pour vous, vous pouvez utiliser la table des matières pour vous guider vers le bon groupe de commandes. Les commandes que nous pensons à la fois particulièrement utiles et faciles à comprendre sont indiquées avec un doigt pointé (☞).

la section 13, "sommaire des commandes", est un index spécialisé qui complète les descriptions plus complètes des sections 5 – 9. Il énumère les commandes de TEX alphabétiquement, avec une brève explication de chaque commande et une référence de la page où il est décrit plus complètement. Le sommaire vous aidera quand vous ne voulez qu'un rappel rapide de la fonction d'une commande.

TEX est un programme complexe qui fonctionne parfois selon sa propre volonté de manière mystérieuse. Dans la section 10, " trucs et astuces", nous fournissons des conseils pour résoudre une variété de problèmes spécifiques que vous pouvez rencontrer de temps en temps. Et si vous êtes dérouté par les messages d'erreur de TEX, vous trouverez de l'aide dans la section 11, "comprendre les messages d'erreur".

Les étiquettes grises sur le côté du livre vous aideront à repérer les parties du livre rapidement. Elles divisent le livre dans les parties principales suivantes :

1) explications générales et exemples
2) concepts
3) descriptions de commandes (cinq étiquettes plus courtes)
4) conseils, messages d'erreur et macros d'`eplain.tex`
5) sommaire des commandes
6) index

Dans beaucoup d'endroits nous avons fourni des références de page de *The TEXbook* (voir la page 18 pour une citation). Ces références s'appliquent à la dix-septième édition de *The TEXbook*. Pour d'autres éditions, quelques références peuvent être dépassées par une ou deux pages.

Conventions syntaxiques

Dans n'importe quel livre concernant la préparation de fichier source pour un ordinateur, il est nécessaire d'indiquer clairement les caractères littéraux qui doivent être saisis et distinguer ces caractères du texte explicatif. Nous employons la police Computer Moderne de machine à écrire pour les `sources littérales comme ceci` ainsi que pour les noms des commandes TEX. Quand il y a possibilité de confusion, nous enfermons le source TEX entre des guillemets simples, '`comme ceci`'. Cependant, nous utilisons de temps en temps des parenthèses quand nous indiquons des caractères simples tels que (') (vous pouvez voir pourquoi).

Pour préserver vos yeux nous ne mettons normalement les espaces que là où vous devez mettre les espaces. Cependant, à quelques endroits où nous devons souligner un espace, nous employons le caractère '␣' pour l'indiquer. Assez naturellement, ce caractère s'appelle un *espace visible*.

Descriptions des commandes

Les sections 5 – 9 contiennent une description de ce que fait presque chaque commande TEX. Les commandes primitives et celles de plain TEX sont couvertes. Les commandes primitives sont celles construites dans le programme TEX, alors que les commandes de plain TEX sont définies dans un fichier standard de définitions auxiliaires (voir p. 88). Les seules commandes que nous avons omises sont celles qui ne sont employées que localement dans la définition de plain TEX (annexe B de *The TEXbook* et de la traduction française). Les commandes sont organisées comme suit :

- "Commandes pour composer des paragraphes", section 5, traite des caractères, des mots, des lignes et des paragraphes entiers.
- "Commandes pour les composer des pages", section 6, traite des pages, de leurs composants et de la routine de rendu.
- "Commandes pour les modes horizontaux et verticaux", section 7, comprend les formes correspondantes ou identiques des modes horizontaux (les paragraphes et les hbox) et les modes verticaux (les pages et les vbox). Ces commandes fournissent les boîtes, les espaces, les règles, les leaders et les alignements.
- "Commandes pour composer les formules de maths", section 8, donne les possibilités de construction de formules mathématique.
- "Commande pour les opérations générales", section 9, fournit les dispositifs de programmation de TEX et tout ce qui n'entre dans aucune des autres sections.

Vous devez voir ces catégories comme étant suggestives plutôt que rigoureuses, parce que les commandes n'entrent pas vraiment de manière ordonnée dans ces (ou tout autre) catégories.

Dans chaque section, les descriptions des commandes sont organisées par fonction. Quand plusieurs commandes sont étroitement liées, elles sont décrites en tant que groupe ; autrement, chaque commande a sa propre explication. La description de chaque commande inclut un ou plusieurs exemples et le rendu produit par chaque exemple quand ils sont appropriés (pour quelques commandes ils ne sont pas). Quand vous regardez une sous- section contenant des commandes fonctionnellement liées, regardez en fin de sous-section l'article "voir aussi" pour vous assurer qu'il ne renvoie pas à des commandes liées qui seraient décrites ailleurs.

Quelques commandes sont étroitement liées à certains concepts. Par exemple, les commandes `\halign` et `\valign` sont liées à "alignement", la commande `\def` est liée à "macro" et les commandes `\hbox` et `\vbox` sont liées à "boîte". Dans ce cas nous avons habituellement donné un squelette de description des commandes elles-mêmes et avons expliqué les idées fondamentales dans le concept.

Les exemples associés aux commandes ont été composés avec une indentation de paragraphe `\parindent`, fixé à zéro pour que les paragraphes soient normalement non indentés. Cette convention facilite la lecture des exemples. Dans les exemples où l'indentation de paragraphe est essentielle, nous l'avons explicitement placée à une valeur différente de zéro.

Le doigt pointé devant une commande ou un groupe de commandes indique que nous jugeons cette commande ou groupe de commandes particulièrement utiles et faciles à comprendre.

Beaucoup de commandes attendent des arguments d'un type ou d'un autre (p. 11). Les arguments d'une commande fournissent à TEX une information complémentaire dont il a besoin afin d'exécuter la commande. Chaque argument est indiqué par un terme entre chevrons imprimé en italique qui indique de quel genre d'argument il s'agit :

⟨*argument*⟩	un seul token ou du texte entre accolades
⟨*code de caractère*⟩	un entier entre 0 et 255
⟨*dimension*⟩	une dimension, c'est-à-dire, une longueur
⟨*ressort*⟩	un ressort (avec étirement et rétrécissement)
⟨*nombre*⟩	un entier éventuellement signé (nombre entier)
⟨*registre*⟩	un numéro de registre entre 0 et 255

Tous ces termes sont expliqués plus en détail dans la section 4. De plus, nous utilisons parfois les termes comme ⟨*liste de token*⟩ qui sont explicites ou expliqués dans la description de la commande. Quelques commandes ont des formats spéciaux qui exigent des accolades ou des mots particuliers. Celles-ci sont placées dans la même police "grasse" que celle utilisée pour les titres de commande.

Quelques commandes sont des paramètres (p. 12) ou des entrées de table. Ceci est indiqué dans le listing de la commande. Vous pouvez employer un paramètre comme argument ou lui assigner une valeur. De

même pour des entrées de table. Nous utilisons le terme "paramètre" pour faire référence aux entités telles que `\pageno` qui sont en fait des registres mais qui se comportent comme des paramètres.

2 Utiliser TEX

Transformer la source en encre

les programmes et fichiers dont vous avez besoin

Afin de produire un document TEX, vous devrez exécuter le programme TEX et ainsi que plusieurs programmes liés. Vous aurez également besoin de fichiers d'aide pour TEX et probablement pour ces autres programmes. Dans ce livre nous ne pouvons vous renseigner qu'au sujet de TEX, mais nous ne pouvons pas vous guider à propos des autres programmes et des fichiers d'aide excepté en des termes très généraux parce qu'ils dépendent de votre environnement TEX local. Les personnes qui vous fournissent TEX devraient pouvoir vous fournir ce que nous appelons de l'*information locale*. L'information locale vous indique comment débuter avec TEX, comment employer les programmes liés et comment accéder aux fichiers d'aide.

Le fichier source TEX se compose d'un fichier texte ordinaire que vous pouvez préparer avec un éditeur de texte. Un fichier source TEX, à la différence d'un dossier source d'un traitement de texte ordinaire, ne contient normalement aucun caractère de contrôle invisible. Vous voyez tout ce que TEX voit si vous regardez le listing du fichier.

Votre fichier source peut s'avérer être un peu plus qu'un squelette qui appelle d'autres fichiers source. Les utilisateurs de TEX organisent souvent de grands documents tels que des livres de cette façon. Vous pouvez utiliser la commande `\input` (p. 255) pour inclure un fichier source dans des autres. En particulier, vous pouvez utiliser `\input` pour inclure des fichiers contenant des *définitions de macro*—définitions auxiliaires qui augmentent les possibilités de TEX. Si des fichiers de macro sont disponibles

sur votre installation TEX, les informations locales sur TEX devraient vous indiquer comment atteindre les fichiers de macro et ce qu'elles peuvent faire pour vous. Le format standard de TEX, celui décrit dans ce livre, inclus une collection de macros et d'autres définitions connus sous le nom de plain TEX (p. 88).

Quand TEX exécute votre document, il produit un fichier appelé fichier `.dvi`. L'abréviation "`dvi`" représente "device independent". L'abréviation a été choisie parce que l'information dans le fichier `.dvi` est indépendante du dispositif que vous utilisez pour imprimer ou afficher votre document.

Pour imprimer votre document ou le regarder avec un *préviewer*, vous devrez traiter le fichier `.dvi` avec un programme de *driver de périphérique*. (Un préviewer est un programme qui vous permet de voir sur un écran une approximation de ce à quoi le résultat composé ressemblera.) Les différents appareils de sortie exigent normalement différents drivers de périphérique. Après avoir exécuté le driver de périphérique, vous devrez également transférer le résultat du driver de périphérique vers l'imprimante ou à tout autre appareil de sortie. Les informations locales sur TEX devraient vous indiquer comment obtenir le driver de périphérique correct et l'utiliser.

Puisque TEX n'a aucune connaissance interne des polices particulières, il emploie des *fichiers de polices* pour obtenir les informations sur les polices utilisées dans votre document. Les fichiers de polices doivent également faire partie de votre environnement TEX local. Chaque police exige normalement deux fichiers : un premier contenant les dimensions des caractères dans la police (le *fichier de métriques*) et un contenant les formes des caractères (le *fichier de formes*). Les versions magnifiées d'une police partagent les métriques mais ont différents dossiers de forme. Les fichiers de métrique sont désignés parfois sous le nom de fichiers `.tfm`, et les différentes variétés de fichiers de forme sous le nom de fichiers `.pk`, fichiers `.pxl` et fichiers `.gf`. Ces noms correspondent aux noms des dossiers que TEX et ses programmes compagnon utilisent. Par exemple, `cmr10.tfm` est le dossier de métrique pour la police `cmr10` (Computer Modern Romain 10 point).

TEX lui-même n'utilise que le fichier des métriques, puisqu'il ne gère pas le dessin des caractères mais seulement l'espace qu'ils occupent. Le driver de périphérique utilise d'habitude le fichier de forme, puisqu'il est responsable de la création de l'image imprimée de chaque caractère composé. Quelques drivers de périphérique doivent aussi utiliser le fichier de métrique. Quelques drivers de périphérique peuvent utiliser les polices résidentes d'une imprimante et n'ont pas besoin des fichiers de forme pour ces polices.

exécuter TEX

Vous pouvez lancer TEX sur un fichier source `screed.tex` en saisissant quelque chose comme '`run tex`' ou juste '`tex`'(vérifiez votre information locale). TEX répondra par quelque chose comme :

```
This is TeX, Version 3.0 (preloaded format=plain 90.4.23)
**
```

Le "format préchargé" ici correspond à une forme prédigérée de macros plain TEX livrée avec TEX. Vous pouvez maintenant saisir '`screed`' pour que TEX traite votre dossier. Quand cela est fait, vous verrez quelque chose comme :

```
(screed.tex [1] [2] [3] )
Output written on screed.dvi (3 pages, 400 bytes).
Transcript written on screed.log.
```

affiché sur votre terminal, ou imprimé dans votre fichier `.log` si vous ne travaillez pas sur un terminal. La majeure partie de ce résultat est explicite. Les nombres entre parenthèses sont des numéros de page que TEX affiche quand il charge chaque page de votre document dans le fichier `.dvi`. TEX mettra normalement une extention '`.tex`' à un nom de fichier source si celui que vous avez donné n'en a pas. Pour quelques formes de TEX vous devez pouvoir appeler TEX directement pour un fichier source en saisissant :

```
tex screed
```

ou quelque chose comme ça.

Au lieu de fournir votre source TEX à partir d'un fichier, vous pouvez le saisir directement sur votre terminal. Pour faire ainsi, saisissez '`\relax`' au lieu de '`screed`' au prompt '`**`'. TEX vous incitera dorénavant avec un '`*`' pour chaque ligne saisie et interprétera chaque ligne saisie comme il la verra. Pour terminer la saisie, tapez une commande telle que '`\bye`' qui indique à TEX que vous avez fini. La saisie directe est parfois une manière pratique d'expérimenter avec TEX.

Quand votre fichier de saisie contient d'autres fichiers de saisie imbriqués, les informations affichées indiquent quand TEX commence et finit de traiter chaque fichier imbriqué. TEX affiche une parenthèse gauche et le nom du fichier quand il commence à travailler sur un fichier et affiche la parenthèse droite correspondante quand il en a fini avec le fichier. Si vous recevez n'importe quels messages d'erreur dans la sortie affichée, vous pouvez les associer avec un fichier en recherchant la parenthèse gauche la plus récemment ouverte.

Pour une explication plus complète de la façon de faire marcher TEX, voir chapitre 6 de *The TEXbook* et de la traduction française et votre information locale.

Préparer un fichier source

Dans cette section nous expliquons certaines des conventions que vous devez suivre en préparant le source pour TEX. Une partie de l'information fournie ici apparaît également dans les exemples dans la section 3 de ce livre.

Commandes et séquences de contrôle

Un source TEX se compose d'une séquence de commandes qui indiquent à TEX comment composer votre document. La plupart des caractères agissent en tant que commandes d'un genre particulièrement simple : "compose-moi". La lettre 'a', par exemple, est une commande pour composer un 'a'. Mais il y a un autre genre de commande—une *séquence de contrôle*— qui donne à TEX des instructions plus raffinées. Une séquence de contrôle commence d'habitude par un antislash (\), bien que vous puissiez changer cette convention si vous en avez besoin. Par exemple, le source :

```
She plunged a dagger (\dag) into the villain's heart.
```

contient la séquence de contrôle `\dag`; elle produit la composition :

She plunged a dagger (†) into the villain's heart.

Tout dans cet exemple excepté le `\dag` et les espaces agissent comme une commande "compose-moi". Nous en expliquerons plus au sujet des espaces dans page 12.

Il y a deux sortes de séquences de contrôle : les *mots de contrôle* et les *symboles de contrôle*

- Un mot de contrôle se compose d'un antislash suivi d'une ou plusieurs lettres, par exemple, '`\dag`'. Le premier caractère qui n'est pas une lettre marque la fin du mot de contrôle.
- Un symbole de contrôle se compose d'un antislash suivi d'un caractère simple qui n'est pas une lettre, par exemple, '`\$`'. le caractère peut être un espace ou même l'extrémité d'une ligne (qui est un caractère parfaitement légitime).

Un mot de contrôle (mais pas un symbole de contrôle) absorbe tous les espaces ou fins de ligne qui le suivent. Si vous ne voulez pas perdre d'espace après un mot de contrôle, faites suivre la commande par un espace contrôlé (\␣) ou par '`{}`'. ainsi :

```
The wonders of \TeX\␣shall never cease!
```

ou :

```
The wonders of \TeX{} shall never cease!
```

produisent :

The wonders of TEX shall never cease!

plutôt que :

The wonders of TEXshall never cease!

qui est ce que vous obtenez si vous enlevez '\␣' ou '{}'.

Ne faite pas suivre un mot de contrôle par le texte qui le suit—TEX ne saura pas où le mot de commande finit. Par exemple, la séquence de contrôle `\c` place une cédille sur le caractère qui le suit. Le mot français *garçon* doit être saisi ainsi '`gar\c␣con`', et non '`gar\ccon `'; si vous écrivez cela, TEX se plaindra au sujet d'une équence de contrôle `\ccon` non définie.

Un symbole de contrôle, par contre, n'absorbe rien de ce qui le suit. Ainsi vous devez saisir '$13.56' ainsi '`\$13.56`', et non '`\$␣13.56`'; la dernière forme produirait '$ 13.56'. Cependant, ces commandes d'accent appelées par des symboles de contrôle sont définies de telle façon qu'elles absorbent l'espace suivant. Ainsi, par exemple, vous pouvez saisir le mot *déshabiller* comme '`d\'eshabiller`' ou comme '`d\'␣eshabiller`'.

Chaque séquence de contrôle est également une commande, mais la proposition inverse n'est pas vraie. Par exemple, la lettre '`N`' est une commande, mais pas une séquence de contrôle. Dans ce livre nous utilisons normalement "commande" plutôt que "séquence de contrôle" quand l'un ou l'autre convient. Nous employons "séquence de contrôle" quand nous voulons souligner les aspects de la syntaxe de TEX qui ne s'appliquent pas aux commandes en général.

Arguments

Quelques commandes doivent être suivies d'un ou plusieurs *arguments* qui aident à déterminer ce que fait la commande. Par exemple, la commande `\vskip`, qui indique à TEX de sauter vers le bas (ou le haut) de la page, s'attend à un argument indiquant de combien d'espace sauter. Pour sauter en bas de deux pouces, vous saisirez '`\vskip 2in`', où `2in` est l'argument de `\vskip`.

Différentes commandes s'attendent à des genres d'arguments différents. Beaucoup de commandes attendent des dimensions, telles que le `2in` dans l'exemple ci-dessus. Quelques commandes, en particulier celles définies par des macros, s'attendent à des arguments qui sont soit un caractère simple ou un texte inclus entre accolades. Pourtant d'autres exigent que leurs arguments soient enfermés entre accolades, c'est-à-dire, qu'elles n'acceptent pas d'arguments sous forme de caractère simple. La description de chaque commande de ce livre vous indique quels genres d'arguments, le cas échéant, la commande attend. Dans certains cas, les accolades exigées définissent un groupe (voir p. 15).

Paramètres

Certaines commandes sont des paramètres (p. 87). Vous pouvez utiliser un paramètre de deux façons :

1) Vous pouvez utiliser la valeur d'un paramètre comme argument d'une autre commande. Par exemple, la commande `\vskip\parskip` provoque un saut vertical de la valeur du paramètre de ressort `\parskip`. (saut de paragraphe)
2) Vous pouvez changer la valeur du paramètre en lui assignant quelque chose. Par exemple, l'assignation `\hbadness=200` met la valeur du nombre du paramètre `\hbadness` à 200.

Nous utilisons aussi le terme "paramètre" pour faire référence aux entités telle que `\pageno` qui sont normalement des registres mais se comporte comme des paramètres.

Certaines commandes sont des noms des tables. Ces commandes sont employées comme des paramètres, sauf qu'elles exigent un argument additionnel qui indique une entrée particulière dans la table. Par exemple, `\catcode` appelle une table de codes de catégorie (p. 56). Ainsi la commande `\catcode‘~=13` place le code de catégorie du caractère ‘~’ à 13.

Espaces

Vous pouvez librement employer les espaces supplémentaires dans votre source. Dans presque toutes les circonstances TEX traite plusieurs espaces d'une rangée comme étant équivalents à un espace simple. Par exemple, il n'importe pas que vous mettiez un ou deux espaces après un point dans votre saisie. Quoique vous fassiez, TEX exécute ses manœuvres de fin de phrase et laisse (dans la plupart des cas) la place appropriée après le point. TEX traite également la fin d'une ligne de source comme équivalente à un espace. Ainsi vous pouvez finir vos lignes de source partout où cela vous arrange—TEX transforme des lignes de source en paragraphes de la même façon quelles que soient les coupures de ligne dans votre source.

Une ligne blanche dans votre source marque la fin d'un paragraphe. Plusieurs lignes blanches sont équivalentes à une seule ligne.

TEX ignore les espaces du source dans les formules mathématiques (voir ci-dessous). Ainsi vous pouvez inclure ou omettre des espaces n'importe où dans une formule mathématique—TEX ne s'en inquiète pas. Même dans une formule mathématique, cependant, vous ne devez pas relier un mot de contrôle avec la lettre qui le suit.

Si vous définissez vos propres macros, vous devez faire particulièrement attention où vous mettez les fins de ligne dans les définitions. Il est très facile de définir une macro qui produit un espace non désiré en plus de ce qu'elle est censée produire. Nous discuterons de ce problème ailleurs parce qu'il est quelque peu technique ; voir page 278.

Un espace ou son équivalent entre deux mots dans votre source ne se transforme pas simplement en caractère d'espace dans votre résultat. Quelques-uns de ces espaces du source se transforment en fins des lignes dans le résultat, puisque, généralement, les lignes du source ne correspondent pas aux lignes du résultat. Les autres se transforment en espaces de largeur variable appelées "ressort" (p. 93), qui ont une taille normale (la taille qu'il "veut avoir") mais qui peut s'étendre ou se rétrécir. Quand TEX compose un paragraphe qui est censé avoir une marge à droite (le cas habituel), il ajuste les largeurs des ressorts de chaque ligne pour obtenir que les lignes de terminent sur la marge. (la dernière ligne d'un paragraphe est une exception, puisqu'on n'exige pas d'habitude qu'elle finisse sur la marge droite.)

Vous pouvez empêcher un espace du source de se transformer en fin de ligne en employant un tilde (`~`). par exemple, vous ne voulez pas que TEX mette une coupure de ligne entre le 'Fig.' et le '8' du 'Fig. 8'. En saisissant '`Fig.~8`' vous pouvez empêcher une telle coupure de ligne.

Commentaires

Vous pouvez insérer des commentaires dans votre source TEXQuand TEX voit un commentaire il passe simplement au-dessus de lui, ainsi ce qui est dans un commentaire n'affecte pas votre document final de quelque façon. Les commentaires sont utiles pour fournir des informations supplémentaires sur le contenu de votre fichier source. Par exemple :

```
% ========= Start of Section 'Hedgehog' =========
```

Un commentaire commence par un signe de pourcentage (`%`) et se prolonge jusqu'à la fin de la ligne du source. TEX n'ignore pas simplement le commentaire mais aussi la fin de la ligne, ainsi des commentaires ont une autre utilisation très importante : relier deux lignes de sorte que la fin de ligne entre elles soit invisible à TEX et ne produise pas d'espace dans le rendu ou de fin de ligne. Par exemple, si vous saisissez :

```
A fool with a spread%
sheet is still a fool.
```

vous obtiendrez :

A fool with a spreadsheet is still a fool.

Ponctuation

TEX ajoute normalement un espace supplémentaire après ce qu'il pense être une marque de ponctuation à la fin d'une phrase, à savoir, '`.`', '`?`' ou '`!`' suivi d'un espace saisi. TEX n'ajoute pas d'espace supplémentaire si le signe de ponctuation suit une lettre majuscule, parce qu'il pense que

la majuscule est l'initiale d'un nom quelconque. Vous pouvez forcer un espace supplémentaire là où il ne se produirait pas autrement en saisissant quelque chose comme :

```
A computer from IBM\null?
```

`\null` ne produit aucun résultat, mais il empêche TEX d'associer la capitale 'M' avec le point d'interrogation. D'autre part, vous pouvez effacer l'espace supplémentaire là où il ne doit pas apparaître en saisissant un espace commandé après le signe de ponctuation, par exemple :

```
Proc.\␣Royal Acad.\␣of Twits
```

permet d'obtenir :

Proc. Royal Acad. of Twits

plutôt que :

Proc. Royal Acad. of Twits

Certains préfèrent ne pas laisser plus d'espace après une ponctuation à la fin d'une phrase. Vous pouvez obtenir cet effet avec la commande `\frenchspacing` (p. 112). `\frenchspacing` est souvent recommandée pour les bibliographies.

Pour les marques de citation simples, vous devez employer les quotes simples gauches et droites (` et ') sur votre clavier. Pour les guillemets doubles gauches et droits, utilisez deux quotes simples gauches ou droites (`` ou '') plutôt que la double quote (") sur votre clavier. La double quote du clavier vous donnera en fait une double marque de citation droite dans beaucoup de polices, mais les deux simples quotes droites sont le style préféré de TEX. Par exemple :

```
There is no `q' in this sentence.
``Talk, child,'' said the Unicorn.
She said, ``\thinspace`Enough!', he said.''
```

Ces trois lignes donnent :

> There is no 'q' in this sentence.
> "Talk, child," said the Unicorn.
> She said, "'Enough!', he said."

`\thinspace` dans la troisième ligne du source empêche la quote simple de venir trop près du guillemet double. Sans lui, vous verriez juste trois guillemets presque équidistants à suivre.

TEX a trois sortes de tirets :

- un court (trait d'union) comme ceci (-). vous l'obtenez en saisissant '`-`'.
- un moyen (demi-cadratin) comme ceci (–). vous l'obtenez en saisissant '`--`'.
- un long (cadratin) comme ceci (—). vous l'obtenez en saisissant '`---`'.

Typiquement, vous utiliserez les traits d'union pour indiquer les mots composés comme "will-o'-the-wisp", les demi-cadratins désignerons les suites de page telles que "pages 81–87", et les cadratins désignerons une coupure dans la continuité—comme ceci.

Caractères spéciaux

Certains caractères ont une signification spéciale en TEX, donc vous ne devez pas les utiliser dans du texte ordinaire. Ce sont :

```
$  #  &  %  _  ^  ~  {  }  \
```

Afin de les produire dans votre document de sortie, vous devez utiliser des circumlocutions. Pour les cinq premiers, vous devez saisir à la place :

```
\$  \#  \&  \%  \_
```

Pour les autres, vous avez besoin de quelque chose de plus raffiné :

```
\^{␣}   \~{␣}   $\{$   $\}$   $\backslash$
```

Groupes

Un *groupe* se compose de code source inclus entre des accolades gauches et droites (`{` et `}`). En plaçant une commande dans un groupe, vous pouvez limiter ses effets au code source compris dans ce groupe. Par exemple, la commande `\bf` indique à TEX de placer quelque chose en **caractères gras**. Si vous deviez mettre `\bf` dans votre source et ne rien faire autrement pour le contrecarrer, tout dans votre document suivant le `\bf` serait placé en caractères gras. En enfermant le `\bf` dans un groupe, vous limitez son effet au groupe. Par exemple, si vous saisissez :

```
We have {\bf a few boldface words} in this sentence.
```

vous obtiendrez :

We have **a few boldface words** in this sentence.

Vous pouvez également utiliser un groupe pour limiter l'effet d'une tâche à un des paramètres de TEX. Ces paramètres contiennent des valeurs qui affectent la façon dont TEX compose votre document. Par exemple, la valeur du paramètre `\parindent` indique l'indentation au début d'un paragraphe. L'assignation `\parindent = 15pt` place l'indentation à 15 points d'imprimeur. En plaçant cette assignation au début d'un groupe contenant plusieurs paragraphes, vous pouvez changer l'impression de ces seuls paragraphes. Si vous n'enfermez pas l'assignation dans un groupe, le changement d'indentation s'appliquera au reste du document (ou jusqu'à la prochaine assignation de `\parindent`, s'il y en a une après).

Toutes les paires d'accolades n'indiquent pas un groupe. En particulier, les accolades associées à un argument pour lequel elles ne sont *pas* exigées

n'indiquent pas un groupe—elles ne servent qu'à délimiter l'argument. De ces commandes qui exigent des accolades pour leurs arguments, certaines traitent les accolades pour définir un groupe et d'autres interprètent l'argument d'une manière spéciale qui dépend de la commande[1].

formules mathématiques

Une formule mathématique peut apparaître dans du *texte mathématique* ou se placer sur une ligne séparée par de l'espace vertical supplémentaire autour d'elle (*mathématiques affichées*). Vous enfermez une formule des textes entre des signes dollar simples (`$`) et une formule affichée entre des signes dollars doubles (`$$`). Par exemple :

```
If $a<b$, then the relation $$e^a < e^b$$ holds.
```

Ce source produit :

If $a < b$, then the relation

$$e^a < e^b$$

holds.

la section 8 décrit les commandes utiles dans les formules mathématiques.

Comment marche TEX

Afin d'utiliser TEX efficacement, il est utile d'avoir une idée de la façon dont TEX organise son activité de transmutation d'entrée en résultat. Vous pouvez imaginer TEX comme une sorte d'organisation avec des "yeux", une "bouche", un "œsophage", un "estomac" et un "intestin". Chaque partie de l'organisation transforme son entrée d'une certaine façon et passe l'entrée transformée à la prochaine étape.

Les yeux transforment un fichier d'entrée en séquence de caractères. La bouche transforme la séquence de caractères en séquence de *tokens*, où chaque token est un caractère simple ou une séquence de contrôle. l'œsophage développe les tokens séquence de *commandes primitives*, qui sont également des tokens. L'estomac effectue les opérations indiquées par les commandes primitives, produisant une séquence de pages. Finalement, l'intestin transforme chaque page dans la forme requise pour le fichier `.dvi` et l'y envoi. Ces actions sont décrites en plus détail dans la section 4 sous "Anatomie de TEX" (p. 48).

[1] plus précisément, pour les commandes primitives, soit les accolades définissent un groupe, soit elles enferment les tokens qui ne sont pas traitées dans l'estomac de TEX. Pour `\halign` et `\valign` le groupe a un effet trivial parce que rien de ce qui est entre accolades n'atteint l'estomac (parce que c'est dans le modèle) ou est inclus dans un nouveau groupe plus profond.

La vraie composition est réalisée dans l'estomac. Les commandes demandent à TEX de composer tel caractère dans telle police, d'insérer de l'espace entre les mots, de finir un paragraphe et ainsi de suite. En commençant par des caractères composés individuellement et d'autres éléments typographiques simples, TEX construit une page comme un ensemble de boîtes de boîtes de boîtes. (voir "boîtes", p. 51). Chaque caractère composé occupe une boîte, et l'ensemble fait une page entière. Une boîte peut contenir des boîtes plus petites mais aussi des *ressorts* (p. 93) et autres diverses choses. Les ressorts produisent de l'espace entre les cases les plus petites. Une propriété importante des ressorts est qu'ils peuvent s'étendre et se rétrécir ; ainsi TEX peut rendre une boîte plus grande ou plus petite en étirant ou en rétrécissant les ressorts.

En général, une ligne est une boîte contenant une suite de boîtes de caractères, et une page est une boîte contenant une suite de boîtes de lignes. Il y a des ressorts entre les mots d'une ligne et entre les lignes d'une page. TEX étire ou rétrécit les ressorts sur chaque ligne afin de justifier une marge droite sur la page et les ressorts de chaque page afin de faire des marges inférieures des différentes pages égales. D'autres genres d'éléments typographiques peuvent également apparaître dans une ligne ou sur une page, mais nous ne les traiterons pas ici.

En tant que processus d'assemblage de pages, TEX à besoin de couper des paragraphes en lignes et des lignes en pages. En effet, l'estomac voit d'abord un paragraphe comme une longue ligne. Il insère des *coupures de ligne* afin de transformer le paragraphe en séquence de lignes de la bonne longueur, exécutant une analyse plutôt raffinée afin de choisir l'ensemble de coupures qui semble la meilleure pour le paragraphe (voir "coupure de ligne", p. 59). L'estomac suit un processus semblable mais plus simple afin de transformer une séquence de lignes en page. Essentiellement l'estomac accumule des lignes jusqu'à ce que plus aucune lignes ne puissent rentrer dans la page. Il choisit alors un endroit simple pour couper la page, en mettant les lignes avant la coupure sur la page courante et en sauvant les lignes après la coupure pour la page suivante (voir "coupure de page", p. 60).

Quand TEX assemble un élément d'une liste d'articles (boîtes, ressorts, etc.), il est dans un des six *modes* (p. 80). Le genre d'entité qu'il assemble définit le mode dans lequel il est. Il y a deux modes ordinaires : le mode horizontal ordinaire pour l'assemblage des paragraphes (avant qu'ils ne soient coupés en lignes) et le mode vertical ordinaire pour les l'assemblage des pages. Il y a deux modes internes : le mode horizontal interne pour assembler des articles horizontalement pour former une boîte horizontale et un mode vertical interne pour assembler des articles verticalement pour former une boîte verticale (autre qu'une page). Finalement, il y a deux modes mathématiques : le mode mathématique de texte pour des formules de maths s'assemblant dans un paragraphe et le mode mathématique d'affichage pour les formules mathématiques qui sont affihées sur des lignes seules (voir "formules mathématiques", p. 16).

Nouveau TEX contre ancien TEX

En 1989, Knuth a fait une révision importante de TEX afin de l'adapter aux jeux de caractères nécessaires à la composition des langues autres que l'anglais.

Cette révision inclus quelques fonctions supplémentaires mineures qui peuvent être ajoutées sans déranger autre chose. Ce livre décrit le "nouveau TEX". Si vous utilisez toujours une version plus ancienne de TEX (version 2.991 ou antérieure), vous devez savoir quels dispositifs du nouveau TEX vous ne pouvez utiliser. Les fonctions suivantes ne sont pas disponibles dans les versions antérieures :

- `\badness` (p. 176)
- `\emergencystretch` (p. 129)
- `\errorcontextlines` (p. 270)
- `\holdinginserts` (p. 155)
- `\language`, `\setlanguage` et `\newlanguage` (pp. 134, 252)
- `\lefthyphenmin` et `\righthyphenmin` (p. 134)
- `\noboundary` (p. 107)
- `\topglue` (p. 162)
- la notation `^^`*xy* pour les chiffres hexadécimaux (p. 54)

Nous vous recommandons de vous procurer le nouveau TEX si vous le pouvez.

Ressources

Un certain nombre de ressources sont disponibles pour vous aider à utiliser TEX. *The TEXbook* est la source définitive d'information sur TEX :

> Knuth, Donald E., *The TEXbook*. Reading, Mass.: Addison-Wesley, 1984.

Assurez-vous de bien obtenir la dix-septième édition (janvier 1990) ou plus; les éditions précédentes ne couvrent pas les dispositifs du nouveau TEX.

LATEX est une collection de commandes très populaire conçues pour simplifier l'utilisation de TEX. Il est décrit dans :

> Lamport, Leslie, *The LATEX Document Preparation System*. Reading, Mass.: Addison-Wesley, 1986.

$\mathcal{AMS}$-TEX est la collection de commandes adoptées par l'American Mathematical Society comme norme pour soumettre de manière électronique des manuscrits mathématiques. Il est décrit dans :

Spivak, Michael D., *The Joy of TEX*. Providence, R.I.: American Mathematical Society, 1986.

Vous pouvez joindre le TEX Users Group (TUG), qui édite un bulletin appelé *TUGBoat*. Le TUG est une excellente source d'informations non seulement sur TEX mais également pour des collections de macros, y compris le $\mathcal{AMS}$-TEX. Son adresse est :

TEX Users Group
c/o American Mathematical Society
P.O. Box 9506
Providence, RI 02940
U.S.A.

En conclusion, vous pouvez obtenir des copies des macros `eplain.tex` décrits dans la section 12 aussi bien que les macros utilisé en composant ce livre. Elles sont disponibles par le réseau Internet par `ftp` anonyme à partir des centres serveurs suivants :

```
labrea.stanford.edu [36.8.0.47]
ics.uci.edu [128.195.1.1]
june.cs.washington.edu [128.95.1.4]
```

La version électronique inclut des macros additionnelles qui composent le source pour le programme machine BIBTEX, écrit par Oren Patashnik à l'université de Stanford, et impriment la sortie de ce programme. Si vous trouvez des bogues dans les macros ou pensez à des améliorations, vous pouvez envoyer un courriel à Karl à `karl@cs.umb.edu`.

Les macros sont également disponibles pour $10.00 US sur des disquettes $5\frac{1}{4}''$ ou $3\frac{1}{2}''$ formatée PC :

Paul Abrahams
214 River Road
Deerfield, MA 01342

Email: `Abrahams%Wayne-MTS@um.cc.umich.edu`

Ces adresses sont correctes pour juin 1990; veuillez prendre en compte qu'elles peuvent changer après cela, en particulier les adresses électroniques.

Ressources en français

[partie spécifique à la version française rajoutée par le traducteur]

Toute la documentation de TEX et consort n'a pas été traduite en français. Néanmoins, de nombreux ouvrages existent dans la langue de Molière. Aussi bien sur TEX et LATEX que sur METAFONT.

TEX

Knuth, Donald E., *Le TEXbook*. Paris, France : Vuibert, 12/2003.

Un grand merci à Jean-Côme Charpentier pour cette traduction tant attendu par la communauTEX francophone.

Seroul, Raymond, *Le petit livre de TEX*. Paris, France : Masson, 1996.

LATEX

Desgraupes, Bernard, *LATEX, apprentissage, guide et référence*. Paris, France : Vuibert, 2000.

très bien fait, un must pour comprendre la gestion des polices sous TEX et LATEX.

Goossens, Michel, *LATEX Companion*. Paris, France : Campus-Press, 2000.

"La" Bible de LATEX, que dire d'autre ? traduction de l'original en anglais, pas très à jour en français mais une nouvelle version anglaise est sortie.

Kluth, Marie-Paule, *FAQ LATEX Française*. Paris, France : Vuibert, 2000.

Très bien fait.

Rolland, Christian, *LATEX. Guide pratique*. Paris, France : O'Reilly, 1999.

très bien aussi, avec le CD-ROM TEXlive en prime (peut-être pas la dernière version mais pratique quand même)

METAFONT

Pour en apprendre un peu plus sur l'univers de TEX.

Desgraupes, Bernard, METAFONT*: guide pratique*. Paris, France : Vuibert, 1999.

pas d'équivalent, à part le METAFONTbook de Knuth (en anglais) .

Vous pouvez joindre l'association GUTenberg (GUTenberg), qui édite un bulletin appelé *La lettre de GUTenberg*. GUTenberg est une excellente source d'informations sur TEX en français. Son adresse est :

TEX Users Group
c/o American Mathematical Society
P.O. Box 9506
Providence, RI 02940
U.S.A.

3 Exemples

Cette section du livre contient un ensemble d'exemples pour vous aider à commencer et pour vous montrer comment faire diverses choses avec TEX. Chaque exemple a le résultat de TEX sur la page de gauche et le source TEX ayant mené à ce résultat sur la page de droite. Vous pouvez utiliser ces exemples comme modèle pour les imiter et comme une manière de trouver les commandes TEX dont vous avez besoin afin de réaliser un effet particulier. Cependant, ces exemples ne peuvent illustrer que quelques-unes des 900 commandes environ de TEX.

Certains de ces exemples sont auto-descriptifs—ainsi, ils décrivent eux-mêmes les dispositifs de TEX qu'ils illustrent. Ces discussions sont nécessairement peu précises parce qu'il n'y a pas la place dans les exemples pour toute l'information dont vous auriez besoin. Le sommaire des commandes (section 13) et l'index vous aideront à localiser l'explication complète de chaque dispositif TEX montrés dans les exemples.

Puisque nous avons conçu les exemples pour illustrer beaucoup de choses à la fois, quelques exemples contiennent une grande variété d'effets typographiques. Ces exemples ne sont généralement pas de bons modèles de pratique typographique. Par exemple, l'exemple 8 a certains de ses numéros d'équation du côté gauche et d'autres du côté droit. Vous ne ferez jamais cela dans une véritable publication.

Chaque exemple, sauf le premier, commence par une macro (voir la p. 73) appelé `\xmpheader`. Nous avons utilisé `\xmpheader` afin de garder de la place dans le source. Sans cela, chaque exemple ferait plusieurs lignes de plus que vous auriez déjà vu. `\xmpheader` produit le titre de l'exemple et l'espace supplémentaire qui va avec. Vous pouvez voir dans le premier exemple ce que fait `\xmpheader`, ainsi vous pouvez l'imiter si vous le souhaitez. A part le `\xmpheader`, chaque commande que nous utilisons dans ces exemples est définie dans plain TEX.

Exemple 1: *Saisir du texte simple*

Il est simple de préparer un texte ordinaire pour TEX puisque TEX n'a pas besoin de savoir comment coupez les lignes dans votre source. Il traite la fin d'une ligne comme un espace†. Si vous ne voulez pas d'espace ici, mettez un signe de pourcentage (le caractère de commentaire) à la fin de la ligne. TEX ignore les espaces en début de ligne, et traite plusieurs espaces comme un seul, même après un point. Vous indiquez un nouveau paragraphe en sautant une ligne (ou plus d'une ligne).

Quand TEX voit un point suivi d'un espace (ou de la fin de la ligne, ce qui est équivalent), il pense normalement que vous avez fini une phrase et insère un petit espace supplémentaire après le point. Il traite les point d'interrogation et d'exclamation de la même manière.

Mais les règles de TEX pour repérer un point ont parfois besoin de réglage. TEX pense qu'une lettre capitale avant un point ne termine pas une phrase, donc vous devez faire quelque chose de légèrement différent si, disons, vous écrivez en parlant d'ADN. C'est une bonne idée de relier des mots ensemble dans des références comme "voir fig. 8" et dans des noms comme V. I. Lénine et dans ... pour que TEX ne puise jamais les couper à un mauvais endroit entre deux lignes. (Les trois points indiquent une ellipse.)

Vous pouvez mettre des citations entre des paires de simples "quotes" gauches et droites pour avoir les bonnes marques de citation doubles gauches et droites. "pour des marques de citation simple et double adjacente, insérez un'espace fin' ". Vous pouvez avoir des tirets demi-cadratins–comme ceci, et des tirets cadratins—comme cela.

† TEX traite aussi une tabulation comme un espace, comme vous le voyez dans ce *pied de page*.

Exemple 1 : Saisir du texte simple **25**

```
% TeX ignores anything on a line after a %
% The next two lines define fonts for the title
\font\xmplbx = cmbx10 scaled \magstephalf
\font\xmplbxti = cmbxti10 scaled \magstephalf
% Now here's the title.
\leftline{\xmplbx Exemple 1:\quad\xmplbxti Saisir du texte simple}
\vglue .5\baselineskip % skip an extra half line
Il est simple de pr\'eparer un texte ordinaire pour \TeX\ puisque
\TeX\ n'a pas besoin de savoir comment coupez les lignes dans
votre source. Il traite la fin d'une ligne comme un espace%
\footnote \dag{\TeX\ traite aussi une tabulation comme un espace, comme
vous le voyez dans ce {\it pied de page}.}.   Si vous ne voulez pas
d'espace ici, mettez un signe de pour%
centage (le caract\'ere de commentaire) \'a la fin de la ligne.
   \TeX\ ignore les espaces en d\'ebut de ligne, et traite
plusieurs          espaces     comme un seul,
m\^eme apr\'es un point.      Vous indiquez un nouveau paragraphe en
sautant une ligne (ou plus d'une ligne).

Quand \TeX\ voit un point suivi d'un espace (ou de la fin de la
ligne, ce qui est \'equivalent), il pense normalement que vous avez
fini une phrase et ins\'ere un petit espace suppl\'ementaire apr\'es le
point.  Il traite les point d'interrogation et d'exclamation de la
m\^eme mani\'ere.

  Mais les r\'egles de \TeX\ pour rep\'erer un point ont parfois besoin
de r\'eglage. \TeX\ pense qu'une lettre capitale avant un point
ne termine pas une phrase, donc vous devez faire quelque chose de
l\'eg\'erement diff\'erent si, disons, vous \'ecrivez en parlant d'ADN\null.
% Le \null emp\'eche TeX de percevoir la capitale `N'
% comme \'etant suivie du point.
C'est une bonne id\'ee de relier des mots ensemble dans des r\'ef\'erences
comme ``voir fig.~8'' et dans des noms comme V.~I\null. L\'enine et dans
$\ldots$ pour que \TeX\ ne puise jamais les couper \'a un mauvais endroit
entre deux lignes.  (Les trois points indiquent une ellipse.)

Vous pouvez mettre des citations entre des paires de simples
``quotes'' gauches et droites pour avoir les bonnes marques de
citation doubles gauches et droites.  ``pour des marques de citation
simple et double adjacente, ins\'erez un`espace fin'\thinspace''. Vous
pouvez avoir des tirets demi-cadratins--comme ceci, et des tirets
cadratins---comme cela.

\bye % end the document
```

Exemple 2 : *Indentation*

Maintenant, regardons comment contrôler l'indentation. Si un traitement de texte sait le faire, TEX le sait sûrement. Notez que ce paragraphe n'est pas indenté

Habituellement, on veut soit indenter les paragraphes, soit laisser de l'espace supplémentaire entre eux. Comme nous n'avons rien changé maintenant, ce paragraphe est indenté.

faisons ces deux paragraphes d'une manière différente, sans indentation et avec six points supplémentaires entre eux.

Voici un autre paragraphe, composé sans indentation. Si nous ne mettons pas d'espace entre ces paragraphes, Nous aurons du mal à savoir où fini l'un et où commence l'autre.

Il est aussi possible d'indenter les deux cotés de paragraphes entiers Les trois paragraphes suivants illustrent cela :

> "Nous avons indenté ce paragraphe des deux cotés d'une indentation de paragraphe. C'est souvent un bon moyen de présenter de longues citations
>
> "Vous pouvez faire plusieurs paragraphes de cette façon si vous le voulez. C'est le second paragraphe indenté simplement."
>
> Vous pouvez même faire des paragraphes doublement indenté si vous en avez besoin. Ceci en est un exemple.

Dans ce paragraphe, nous revenons aux marges normales, comme vous pouvez le voir. Rallongeons la sauce pour que les marges soient clairement visibles.

> Maintenant, nous indentons la marge de gauche d'un demi pouce et laissons la marge droite à sa position usuelle

Finalement, nous indentons la marge droite d'un demi pouce et laissons la marge gauche à sa position usuelle.

Exemple 2 : Indentation **27**

```
\xmpheader 2/{Indentation}% voir p. 23
\noindent Maintenant, regardons comment contr\^oler l'indentation.
Si un traitement de texte sait le faire, \TeX\ le sait s\^urement.
Notez que ce paragraphe n'est pas indent\'e

Habituellement, on veut soit indenter les paragraphes, soit
laisser de l'espace suppl\'ementaire entre eux. Comme nous n'avons
rien chang\'e maintenant, ce paragraphe est indent\'e.

{\parindent = 0pt \parskip = 6pt
% l'accolade gauche d\'ebute un groupe contenant le texte non indent\'e
faisons ces deux paragraphes d'une mani\`ere diff\'erente,
sans indentation et avec six points suppl\'ementaires
entre eux.

Voici un autre paragraphe, compos\'e sans indentation. Si nous
ne mettons pas d'espace entre ces paragraphes, Nous aurons du mal
\`a savoir o\`u fini l'un et o\`u commence l'autre.
\par % Le paragraphe *doit* \^etre termin\'e dans le groupe.
}% L'accolade droite termine le groupe contenant du texte nin indent\'e.

Il est aussi possible d'indenter les deux cot\'es de paragraphes entiers
Les trois paragraphes suivants illustrent cela~:
\smallskip % Procure un peu d'espace suppl\'ementaire ici.
% Skips like this and \vskip below end a paragraph.
{\narrower
``Nous avons indent\'e ce paragraphe des deux cot\'es d'une indentation de
paragraphe. C'est souvent un bon moyen de pr\'esenter de longues citations

``Vous  pouvez faire plusieurs paragraphes de cette fa\c con si vous
le voulez. C'est le second paragraphe indent\'e simplement.''\par}

{\narrower \narrower Vous pouvez m\^eme faire des paragraphes
doublement indent\'e si vous en avez besoin. Ceci en est un
exemple.\par}
\vskip 1pc % Saut vers le bas d'un pica pour s\'eparation visuelle.
Dans ce paragraphe, nous revenons aux marges normales, comme vous
pouvez le voir. Rallongeons la sauce pour que les marges soient
clairement visibles.

{\leftskip .5in Maintenant, nous indentons la marge de gauche d'un
demi pouce et laissons la marge droite \`a sa position usuelle\par}
{\rightskip .5in Finalement, nous indentons la marge droite d'un
demi pouce et laissons la marge gauche \`a sa position usuelle.\par}
\bye % end the document
```

Exemple 3 : *Polices et caractères spéciaux*

Voici quelques mots en police italique, **quelques autres en police grasse**, *et un* **mélange** *des deux, avec deux* mots en romain *insérés*. Quand une police italique est suivie par une non italique, on insère une "correction d'italique" (\/) pour que l'espacement rende bien. Voici un mot plus petit—mais les polices standard de TEX ne vous en donnerons pas de plus petit de celui-là.

Si vous voulez un de ces dix caractères :

$ & # _ % ^ ~ { } \

vous devez les écrire d'une manière spéciale. Regardez la page d'en face pour savoir comment. TEX a les accents et les lettres dont vous avez besoin pour des mots Français comme *rôle* et *élève*, pour des mots Allemands comme *Schuß* aussi bien que pour des mots de plein d'autres langues. Vous trouverez une liste complète des accents de TEX et des lettres de langues Européennes à la page 103 et à la page 106.

Vous trouverez aussi des lettres Grecques comme "α" et "Ω" pour les utilisez en mathématique, des couleurs de carte comme "♠" et "♢", des symboles de musique comme "♯" et "♭", et plein d'autres symboles spéciaux dont vous trouverez la liste à la page 196. TEX n'acceptera ces sortes de symboles spéciaux que dans son "mode mathématique", donc vous devez les entourer entre caractères '$'.

Exemple 3 : Polices et caractères spéciaux **29**

```
\xmpheader 3/{Polices et caract\‘eres sp\’eciaux}% voir p. 23
\chardef \\ = ‘\\ % Let \\ denote a backslash.
{\it Voici quelques mots en police italique}, {\bf quelques
autres en police grasse}, {\it et un\/ {\bf m\’elange}
des deux, avec deux\/ {\rm mots en romain} ins\’er\’es}.
Quand une police italique est suivie par une non italique, on
ins\‘ere une ‘‘correction d’italique’’ ({\tt \\/}) pour que
l’espacement rende bien.
Voici un mot plus {\sevenrm petit}---mais les polices standard
de \TeX\ ne vous en donnerons pas de plus petit de {\fiverm celui-l\‘a}.

Si vous voulez un de ces dix caract\‘eres~:
\medskip
\centerline{\$ \quad \& \quad \# \quad \_ \quad \% \quad
   \char ‘\^ \quad \char ‘\~ \quad $\{$ \quad
   $\}$ \quad $\backslash$}
% Le \quad ins\‘ere un espace cadratin entre les caract\‘eres.
\medskip
\noindent  vous devez les \’ecrire d’une mani\‘ere sp\’eciale. Regardez
la page d’en face pour savoir comment.
\TeX\ a les accents et les lettres dont vous avez besoin
pour des mots Fran\c cais comme {\it r\^ ole\/} et  {\it \’
el\‘ eve\/}, pour des mots Allemands comme {\it Schu\ss\/}
aussi bien que pour des mots de plein d’autres langues.
Vous trouverez une liste compl\‘ete des accents de \TeX\ et des lettres
de langues Europ\’eennes \‘a la page 103 et \‘a la page 106.

Vous trouverez aussi des lettres Grecques comme ‘‘$\alpha$’’ et
‘‘$\Omega$’’ pour les utilisez en math\’ematique, des couleurs de carte
comme ‘‘$\spadesuit$’’ et ‘‘$\diamondsuit$’’, des symboles de musique
comme ‘‘$\sharp$’’ et ‘‘$\flat$’’, et plein d’autres symboles
sp\’eciaux dont vous trouverez la liste \‘a la page 196.
\TeX\ n’acceptera ces sortes de symboles sp\’eciaux que dans son
‘‘mode math\’ematique’’, donc vous devez les entourer
entre caract\‘eres ‘{\tt \$}’.
\bye % end the document
```

Exemple 4 : ***Espacement interligne***

Un jour, vous voudrez imprimer un document avec de l'espace supplémentaire entre les lignes. Par exemple, les actes du congrès sont imprimés ainsi pour que les législateurs puissent les annoter. Pour la même raison, les éditeurs insistent habituellement pour que les manuscrits ait un espacement double. L'espacement double est rarement approprié pour les documents finis, néanmoins.

Une ligne de base est une ligne imaginaire qui agit comme les lignes horizontales d'un papier quadrillé. Vous pouvez contrôler l'espacement interligne—ce que les imprimeurs appellent "interlignage"—en fixant la taille de l'espace entre les lignes de base. regardez le source pour voir comment faire. Vous pouvez utiliser la même méthode pour un espacement de 1 1/2, en mettant `1.5` au lieu de `2`. (Vous pouvez aussi écrire 1½ d'une meilleur façon.)

Pour cet exemple, nous avons aussi augmenté l'indentation de paragraphe et sauté une ligne de plus entre les paragraphes.

```
\xmpheader 4/{Espacement interligne}% voir p. 23
\baselineskip = 2\baselineskip % double spacing
\parskip = \baselineskip % Skip a line between paragraphs.
\parindent = 3em % Increase indentation of paragraphs.

% The following macro definition gives us nice inline
% fractions.  You'll find it in our eplain macros.
\def\frac#1/#2{\leavevmode
   \kern.1em \raise .5ex \hbox{\the\scriptfont0 #1}%
   \kern-.1em $/$%
   \kern-.15em \lower .25ex \hbox{\the\scriptfont0 #2}%
}%

Un jour, vous voudrez imprimer un document avec de l'espace
suppl\'ementaire entre les lignes. Par exemple, les actes du congr\`es
sont imprim\'es ainsi pour que les l\'egislateurs puissent les annoter.
Pour la m\^eme raison, les \'editeurs insistent habituellement pour que
les manuscrits ait un espacement double. L'espacement double est
rarement appropri\'e pour les documents finis, n\'eanmoins.

Une ligne de base est une ligne imaginaire qui agit comme les
lignes horizontales d'un papier quadrill\'e. Vous pouvez contr\^oler
l'espacement interligne---ce que les imprimeurs appellent
``interlignage''---en fixant la taille de l'espace entre les lignes de
base. regardez le source pour voir comment faire. Vous pouvez
utiliser la m\^eme m\'ethode pour un espacement de $1\;1/2$, en mettant
{\tt 1.5} au lieu de {\tt 2}. (Vous pouvez aussi \'ecrire $1\frac 1/2$
d'une meilleur fa\c con.)
% Here we've used the macro definition given above.

Pour cet exemple, nous avons aussi augment\'e l'indentation de
paragraphe et saut\'e une ligne de plus entre les paragraphes.
\bye % end the document
```

Exemple 5 : *Espacement, traits et boîtes*

Voici un exemple de "liste descriptive". En pratique vous feriez mieux de creer une macro pour avoir des constructions répétitive et être sur que la largeur des sous-titres soit suffisament grande :

La reine de Cœur Une femme à moitié folle, prompte a déclarer "coupez-lui la tête !" à la moindre provocation.

Le chat de Cheshire Un chat avec un énorme sourire qu'Alice trouve dans un arbre.

La tortue Mock une créature larmoyante, un peu menteuse, qui était un compagnon du Gryphon. Réputé être l'ingrédient principal de la soupe de tortue Mock.

Voici un exemple de mots dans une boîte tracée, comme Lewis Carroll l'a écrite :

Who would not give all else for twop ennyworth only of Beautiful Soup?

* * * * * * * * * * * * * * * * * * *

Ici, nous obtenons l'effet d'une barre de révision sur le texte de ce paragraphe. La barre de révision indique un changement.

```
\xmpheader 5/{Espacement, traits et bo\^\i tes}% voir p. 23
Voici un exemple de ``liste descriptive''. En pratique vous
feriez mieux de creer une macro pour avoir des constructions r\'ep\'etitive
et \^etre sur que la largeur des sous-titres soit suffisament grande~:
\bigskip
% Call the indentation for descriptions \descindent
% and set it to 8 picas.
\newdimen\descindent \descindent = 9pc
% Indent paragraphs by \descindent.
% Skip an additional half line between paragraphs.
{\noindent \leftskip = \descindent  \parskip = .5\baselineskip
% Move the description to the left of the paragraph.
\llap{\hbox to \descindent{\bf La reine de C\oe ur\hfil}}%
Une femme \`a moiti\'e folle, prompte a d\'eclarer ``coupez-lui
la t\^ete~!''\ \`a la moindre provocation.\par
\noindent\llap{\hbox to \descindent{\bf Le chat de Cheshire\hfil}}%
Un chat avec un \'enorme sourire qu'Alice trouve
dans un arbre.\par
\noindent\llap{\hbox to \descindent{\bf La tortue Mock\hfil}}%
une cr\'eature larmoyante, un peu menteuse, qui \'etait un
compagnon du Gryphon.  R\'eput\'e \^etre l'ingr\'edient principal
de la soupe de tortue Mock.
\par}
\bigskip\hrule\bigskip % A line with vertical space around it.
Voici un exemple de mots dans une bo\^\i te trac\'ee, comme
Lewis Carroll l'a \'ecrite~:
\bigskip
% Put 8pt of space between the text and the surrounding rules.
\hbox{\vrule\vbox{\hrule
   \hbox spread 8pt{\hfil\vbox spread 8pt{\vfil
      \hbox{Who would not give all else for twop}%
      \hbox{ennyworth only of Beautiful Soup?}%
   \vfil}\hfil}
\hrule}\vrule}%

\bigskip\line{\hfil\hbox to 3in{\leaders\hbox{ * }\hfil}\hfil}
\bigskip

\line{\hskip -4pt\vrule\hfil\vbox{
Ici, nous obtenons l'effet d'une barre de r\'evision sur le texte
de ce paragraphe. La barre de r\'evision indique un changement.}}
\bye % end the document
```

Exemple 6 : *Odds and ends*

TEX sait couper les mots, mais n'est pas infaillible. Si vous discutez de l'ingrédient chimique 5-[p-(Flourosulfonyl)benzoyl]-l,N^6-ethenoadenosine et que TEX se plaint auprès de vous d'un "overfull hbox", essayez d'insérer des "césures discrétionnaires". La notation '\-' indique à TEX les césures discrétionnaire, qui sont celles qui n'auraient pas été inséré autrement.

Vous pouvez composer une texte non justifié, c'est-à-dire, avec une marge droite non alignée. Autrefois, avant que mes traitements de texte soient commun, les document tapés à la machine parce qu'il n'y avait pas de solutions pratiques. Certaines personnes préfèrent que le texte ne soit pas justifié parce que l'espacement entre les mots est uniforme. La plupart des livres sont fait avec des marges justifiées, mais pas tous.

Assertion 27. *Il y a un moyen simple de composer les en-têtes d'assertions, lemmes, théorèmes, etc.*

Voici un exemple de comment composer une liste d'items à deux niveaux de profondeur. Si vous avez besoin de niveau supplémentaires, vous devrez les programmer vous-même, hélas.

1. Voici le premier item.
2. Voici le second item. Il est constitué de deux paragraphes. Nous avons indenté le second paragraphe pour que vous puissiez voir facilement où il commence.

   Le second paragraphe a trois sous-items sous lui.

   (a) Voici le premier sous-item.
   (b) Voici le second sous-item.
   (c) Voici le troisième sous-item.

- Ceci est un étrange item parce qu'il est complètement différent des autres.

Voici une ligne justifiée à gauche.⇐

⇒Voici une ligne justifiée à droite.

⇒Voici une ligne centrée.⇐

Exemple 6 : Odds and ends **35**

```
\xmpheader 6/{Odds and ends}% voir p. 23
\chardef \\ = `\\ % Let \\ denote a backslash.
\footline{\hfil{\tenit - \folio -}\hfil}
% \footline provides a footer line.
% Here it's a centered, italicized page number.
\TeX\ sait couper les mots, mais n'est pas infaillible.
Si vous discutez de l'ingr\'edient chimique
${\it 5}$-[p-(Flouro\-sul\-fonyl)ben\-zoyl]-l,%
$N^6$-ethe\-no\-adeno\-sine
et que \TeX\ se plaint aupr\`es de vous d'un ``overfull hbox'', essayez
d'ins\'erer des ``c\'esures discr\'etionnaires''. La notation
`{\tt \\-}' indique \`a \TeX\ les c\'esures  dis\-cr\'e\-tionnaire,
qui sont celles qui n'auraient pas \'et\'e ins\'er\'e autrement.
\medskip
{\raggedright  Vous pouvez composer une texte non justifi\'e, c'est-\`a-dire,
avec une marge droite non align\'ee. Autrefois, avant que mes traitements de
texte soient commun, les document tap\'es \`a la machine parce qu'il n'y avait
pas de solutions pratiques.
Certaines personnes pr\'ef\`erent que le texte ne soit pas justifi\'e
parce que l'espacement entre les mots est uniforme. La plupart des
livres sont fait avec des marges justifi\'ees, mais pas tous. \par}

\proclaim Assertion 27. Il y a un moyen simple de composer
les en-t\^etes d'asser\-tions, lemmes, th\'eor\`emes, etc.

Voici un exemple de comment composer une liste d'items \`a deux
niveaux de profondeur.  Si vous avez besoin de niveau suppl\'ementaires,
vous devrez les programmer vous-m\^eme, h\'elas.
\smallskip
\item {1.} Voici le premier item.
\item {2.} Voici le second item.  Il est constitu\'e de deux
paragraphes.  Nous avons indent\'e le second paragraphe pour que
vous puissiez voir facilement o\`u il commence.

\item{} \indent Le second paragraphe a trois sous-items
sous lui.
\itemitem {(a)} Voici le premier sous-item.
\itemitem {(b)} Voici le second sous-item.
\itemitem {(c)} Voici le troisi\`eme sous-item.
\item {$\bullet$} Ceci est un \'etrange item parce qu'il est
compl\`etement diff\'erent des autres.
\smallskip
\leftline{Voici une ligne justifi\'ee \`a gauche.$\Leftarrow$}
\rightline{$\Rightarrow$Voici une ligne justifi\'ee \`a droite.}
\centerline{$\Rightarrow$Voici une ligne centr\'ee.$\Leftarrow$}
% Don't try to use these commands within a paragraph.
\bye % end the document
```

Exemple 7 : ***Utiliser des polices venant d'ailleurs***

Vous n'êtes pas restreint à la police Computer Modern qui est fournie avec TEX. D'autres polices sont possibles provenant d'autres sources et vous pouvez les préférer. Par exemple, nous avons composé cette page en Palatino Romain 10 points. Palatino a été dessinée par Hermann Zapf, considéré comme un des plus grand typographe du vingtième siècle. Cette page vous donnera une idée de ce à quoi elle ressemble.

Les polices peuvent provenir soit sous forme vectorielles soit de bitmaps. une police vectorielle décrit le dessin des caractères, tandis qu'une bitmap spécifie chaque pixel (point) qui forme chaque caractère. Une police vectorielle peut être utilisée pour générer différentes tailles de la même police. Le programme Metafont qui est associé à TEX procure un moyen particulièrement puissant de générer des polices bitmap, mais ce n'est pas le seul moyen.

Le fait qu'une seule routine puisse générer un grand éventail de tailles de point pour une police tente beaucoup de vendeurs de polices digitales de ne produire qu'une taille de vectorielles pour une police comme Palatino Romain. Cela semble une décision économiquement sensible, mais c'est un sacrifice esthétique. Une police ne peut pas être agrandie et rétrécie linéairement sans perdre en qualité. Les grandes lettres ne doivent pas, en général, avoir les mêmes proportions que les plus petites ; Cela ne rend simplement pas beau. Par exemple, une police qui est diminuée linéairement aura trop peu d'espace entre ses lettres et la hauteur des minuscules sera trop petite.

Un typographe peut compenser ces changements en procurant différentes vectorielles pour différentes tailles de point, mais il est nécessaire de dépenser du temps à dessiner ces différentes vectorielles. Un des grands avantages de Metafont est qu'il est possible de paramétrer les descriptions des caractères dans une police. Metafont peut alors maintenir la qualité typographique des caractères à travers une échelle de taille de point en ajustant le dessin des caractères en conséquence.

Exemple 7 : Utiliser des polices venant d'ailleurs **37**

```
\xmpheader 7/{Utiliser des polices venant d'ailleurs}% voir p. 23
\font\tenrm = pplr % Palatino
%\font\tenrm = pnss10 % lucida
% Define a macro for invoking Palatino.
\def\pal{\let\rm = \tenrm \baselineskip=12.5pt \rm}
\pal % Use Palatino from now on.

Vous n'\^etes pas restreint \'a la police Computer Modern qui est
fournie avec \TeX. D'autres polices sont possibles provenant d'autres
sources et vous pouvez les pr\'ef\'erer. Par exemple, nous avons
compos\'e cette page en Palatino Romain 10 points. Palatino a \'et\'e
dessin\'ee par Hermann Zapf, consid\'er\'e comme un des plus grand
typographe du vingti\'eme si\'ecle. Cette page vous donnera une id\'ee
de ce \'a quoi elle ressemble.

Les polices peuvent provenir soit sous forme vectorielles soit de
bitmaps. une police vectorielle d\'ecrit le dessin des caract\'eres,
tandis qu'une bitmap sp\'ecifie chaque pixel (point) qui forme chaque
caract\'ere. Une police vectorielle peut \^etre utilis\'ee pour
g\'en\'erer diff\'e\-rentes tailles de la m\^eme police. Le programme
Metafont qui est associ\'e \'a \TeX\ procure un moyen
particuli\'erement puissant de g\'en\'erer des polices bitmap, mais ce
n'est pas le seul moyen.

Le fait qu'une seule routine puisse g\'en\'erer un grand \'eventail de
tailles de point pour une police tente beaucoup de vendeurs de polices
digitales de ne produire qu'une taille de vectorielles pour une police
comme Palatino Romain. Cela semble une d\'ecision \'economiquement
sensible, mais c'est un sacrifice esth\'etique. Une police ne peut pas
\^etre agrandie et r\'etr\'ecie lin\'eairement sans perdre en
qualit\'e. Les grandes lettres ne doivent pas, en g\'en\'eral, avoir
les m\^emes proportions que les plus petites~; Cela ne rend simplement
pas beau. Par exemple, une police qui est diminu\'ee lin\'eairement
aura trop peu d'espace entre ses lettres et la hauteur des minuscules
sera trop petite.
%For example, a font that's linearly scaled down will
%tend to have too little space between strokes, and its
%x-height will be too~small. % tie added to avoid widow word

Un typographe peut compenser ces changements en procurant
diff\'e\-rentes vectorielles pour diff\'erentes tailles de point, mais
il est n\'ecessaire de d\'epenser du temps \'a dessiner ces
diff\'erentes vectorielles. Un des grands avantages de Metafont est
qu'il est possible de param\'etrer les descriptions des caract\'eres
dans une police. Metafont peut alors maintenir la qualit\'e
typographique des caract\'eres \'a travers une \'echelle de taille de
point en ajustant le dessin des caract\'eres en cons\'equence.
\bye % end the document
```

Exemple 8 : ***Un tableau tracé***

Quelques Champignons Remarquables

Nom Botanique	**Nom Commun**	**Caractèristiques d'Identification**
Pleurotus ostreatus	Pleurote en forme d'huitre	Poussent étagés les uns sur les autres sur le bois mort ou affaibli, chapeau en forme de coquille d'huitre gris-rose, pied court ou absent.
Lactarius hygrophoroides	Lactaire hygrophore	Chapeau et pied de couleur orange rouille, abondance de lait, souvent sur la terre des bois près des ruisseaux.
Morchella esculenta	Morille commune	Chapeau conique avec des trous noirs et les arêtes blanches; sans lamelles. Souvent près de vieux pommiers et des ormes morts au printemps.
Boletus edulus	Cèpe de Bordeaux	Chapeau rouge-brun à café au lait avec des pores jaunes (blanc quand il est jeune), pied bulbeux, souvent près des conifères, bouleaux ou trembles.

Exemple 8 : Un tableau tracé **39**

```
\xmpheader 8/{Un tableau trac\'e}% voir p. 23
\bigskip
\offinterlineskip % So the vertical rules are connected.
% \tablerule constructs a thin rule across the table.
\def\tablerule{\noalign{\hrule}}
% \tableskip creates 9pt of space between entries.
\def\tableskip{\omit&height 9pt&&&\omit\cr}
% & separates templates for each column. TeX substitutes
% the text of the entries for #. We must have a strut
% present in every row of the table; otherwise, the boxes
% won't butt together properly, and the rules won't join.
\halign{\tabskip = .7em plus 1em  % glue between columns
% Use \vtop for short multiline entries in the first column.
% Typeset the lines ragged right, without hyphenation.
   \vtop{\hsize=6pc\pretolerance = 10000\hbadness = 10000
      \normalbaselines\noindent\it#\strut}%
  &\vrule #&#\hfil &\vrule #% the rules and middle column
% Use \vtop to get whole paragraphs in the last column.
  &\vtop{\hsize=11pc \parindent=0pt \normalbaselineskip=12pt
    \normalbaselines \rightskip=3pt plus2em #}\cr
% The table rows begin here.
\noalign{\hrule height2pt depth2pt \vskip3pt}
  % The header row spans all the columns.
  \multispan5\bf Quelques Champignons Remarquables\hfil\strut\cr
\noalign{\vskip3pt} \tablerule
  \omit&height 3pt&\omit&&\omit\cr
  \bf Nom&&\bf Nom &&\omit \bf Caract\'eristiques \hfil\cr
\noalign{\vskip -2pt}% close up lines of heading
  \bf Botanique&&\bf Commun&&\omit \bf d'Identification \hfil\cr
\tableskip Pleurotus ostreatus&&Pleurote en forme d'huitre&&
  Poussent \'etag\'es les uns sur les autres sur le bois mort ou affaibli,
  % without the kern, the `f' and `l' would be too close
  chapeau en forme de coquille d'huitre gris-rose, pied court ou absent.\cr
\tableskip Lactarius hygrophoroides&&Lactaire hygrophore&&
  Chapeau et pied de couleur orange rouille, abondance de lait,
  souvent sur la terre des bois pr\'es des ruisseaux.\cr
\tableskip Morchella esculenta&&Morille commune&&Chapeau conique
  avec des trous noirs et les ar\^etes blanches; sans lamelles. Souvent
  pr\'es de vieux pommiers et des ormes morts au printemps.\cr
\tableskip Boletus edulus&&C\'epe de Bordeaux&&Chapeau rouge-brun \'a
  caf\'e au lait avec des pores jaunes (blanc quand il est jeune),
  pied bulbeux, souvent pr\'es des conif\'eres, bouleaux ou~trembles.\cr
\tableskip \tablerule \noalign{\vskip 2pt} \tablerule
}\bye
```

Exemple 9 : ***Composer des mathématiques***

Pour un triangle sphérique de cotés a, b et c et d'angles opposés α, β et γ, nous avons :

$$\cos\alpha = -\cos\beta\cos\gamma + \sin\beta\sin\gamma\cos\alpha \quad \text{(Loi de Cosinus)}$$

et :

$$\tan\frac{\alpha}{2} = \sqrt{\frac{-\cos\sigma\cdot\cos(\sigma-\alpha)}{\cos(\sigma-\beta)\cdot\cos(\sigma-\gamma)}}, \quad \text{où } \sigma = \tfrac{1}{2}(a+b+c)$$

Nous avons aussi :

$$\sin x = \frac{e^{ix} - e^{-ix}}{2i}$$

et :

$$\int_0^\infty \frac{\sin ax \sin bx}{x^2}\,dx = \frac{\pi a}{2}, \quad \text{si } a < b$$

Le nombre de combinaisons ${}_nC_r$ de n choses prises parmi r est :

$$C(n,r) = {}_nC_r = \binom{n}{r} = \frac{n(n-1)\cdots(n-r+1)}{r(r-1)\cdots(1)} = \frac{n!}{r!(n-r)!}$$

La valeur du déterminant D d'ordre n :

$$D = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}$$

est définie comme la somme de $n!$ termes :

$$\sum (\pm)\, a_{1i}a_{2j}\dots a_{nk}$$

où i, j, …, k prenent toutes les valeurs possibles entre 1 et n et pour lequel, le signe du produit est $+$ si la séquence i, j, …, k est une permutation paire et $-$ autrement. De plus :

$$Q(\xi) = \lambda_1 y_1^2 \sum_{i=2}^{n}\sum_{j=2}^{n} y_i b_{ij} y_j, \qquad B = \|b_{ij}\| = B'$$

```
\xmpheader 9/{Composer des math\'ematiques}% voir p. 23
Pour un triangle sph\'erique de cot\'es $a$, $b$ et $c$ et
d'angles oppos\'es $\alpha$, $\beta$ et $\gamma$, nous avons~:
$$\cos \alpha = -\cos \beta \cos \gamma +
  \sin \beta \sin \gamma \cos \alpha \quad
  \hbox{(Loi de Cosinus)}$$
et~:
$$\tan {\alpha \over 2} = \sqrt{
  {- \cos \sigma \cdot \cos(\sigma - \alpha)} \over
  {\cos (\sigma - \beta) \cdot \cos (\sigma - \gamma)}},\quad
  \hbox{o\`u $\sigma = {1 \over 2}(a+b+c)$}$$
Nous avons aussi~:$$\sin x = {{e^{ix}-e^{-ix}}\over 2i}$$
et~:
$$\int _0 ^\infty {{\sin ax \sin bx}\over{x^2}}\,dx
% The \, above produces a thin space
  = {\pi a\over 2}, \quad \hbox{si $a < b$}$$

\noindent Le nombre de combinaisons ${}_nC_r$ de $n$
choses prises parmi $r$ est~:
$$C(n,r) = {}_nC_r = {n \choose r} =
  {{n(n-1) \cdots (n-r+1)} \over {r(r-1) \cdots (1)}} =
  {{n!}\over {r!(n-r)!}}$$

\noindent
La valeur du d\'eterminant $D$ d'ordre $n$~:
$$D = \left|\matrix{a_{11}&a_{12}&\ldots&a_{1n}\cr
  a_{21}&a_{22}&\ldots&a_{2n}\cr
  \vdots&\vdots&\ddots&\vdots\cr
  a_{n1}&a_{n2}&\ldots&a_{nn}\cr}\right| $$
est d\'efinie comme la somme de $n!$ termes~:
$$\sum\>(\pm)\>a_{1i}a_{2j} \ldots a_{nk}$$
% The \> above produces a medium space.
o\`u $i$, $j$, \dots,~$k$\/ prenent toutes les valeurs possibles
entre $1$ et $n$ et pour  lequel, le signe du produit est
$+$ si la s\'equence $i$, $j$, \dots,~$k$\/ est une
permutation paire et $-$ autrement.  De plus~:
$$Q(\xi) = \lambda_1 y_1^2 \sum_{i=2}^n \sum_{j=2}^n y_i
b_{ij} y_j,\qquad B = \Vert b_{ij} \Vert = B'$$
\bye
```

Exemple 10 : *Plus de mathématiques*

La valeur absolue de X, $|x|$, est définie par :

$$|x| = \begin{cases} x, & \text{si } x \geq 0; \\ -x, & \text{autrement.} \end{cases}$$

Maintenant quelques équations numérotées. Il arrive que pour $k \geq 0$:

$$x^{k^2} = \overbrace{x\, x\, \cdots\, x}^{2k \text{ fois}} \tag{1}$$

Voici un exemple qui montre certains controles d'espacement, avec un numéro à gauche :

$$[u][v][w]\,[x]\,[y]\,[z] \tag{2a}$$

Le montant d'espace entre les items entre crochets augmente graduellement de gauche à droite. (Nous avons fait l'espace entre les deux premiers items plus *petit* que l'espace naturel.) Il arrive parfois que

$$u'_1 + tu''_2 = u'_2 + tu''_1 \tag{2b}$$

$$\hat{\imath} \neq \hat{\jmath} \tag{2c}$$

$$\vec{a} \approx \vec{b}$$

Le résultat est d'ordre $O(n \log\log n)$. D'où

$$\sum_{i=1}^{n} x_i = x_1 + x_2 + \cdots + x_n = \mathrm{Sum}(x_1, x_2, \ldots, x_n). \tag{3}$$

et

$$dx\, dy = r\, dr\, d\theta. \tag{4}$$

Le jeu de tous les q tels que $q \leq 0$ s'écrit :

$$\{\, q \mid q \leq 0 \,\}$$

Donc

$$\forall x \exists y\ P(x,y) \Rightarrow \exists x \exists y\ P(x,y)$$

où

$$P(x,y) \stackrel{\text{def}}{\equiv} \text{tout prédicat en } x \text{ et } y.$$

```
\xmpheader 10/{Plus de math\'ematiques}% voir p. 23
La valeur absolue de $X$, $|x|$, est d\'efinie par~:
$$|x| = \cases{x, &si $x\ge 0$;\cr
-x,&autrement.\cr}$$
Maintenant quelques \'equations num\'erot\'ees.
Il arrive que pour $k \ge 0$~:
$$x^{k^2}=\overbrace{x\>x\>\cdots\> x}^{2k\ \rm fois}
   \eqno (1)$$

Voici un exemple qui montre certains controles d'espacement, avec
un num\'ero  \`a gauche~:
$$[u]\![v][w]\,[x]\>[y]\;[z]\leqno(2a)$$
Le montant d'espace entre les items entre crochets
augmente graduellement de gauche \`a droite.  (Nous avons fait
l'espace entre les deux premiers items plus {\it petit\/}
que l'espace naturel.)
Il arrive parfois que $$\leqalignno{
u'_1 + tu''_2 &= u'_2 + tu''_1&(2b)\cr
\hat\imath &\ne \hat \jmath&(2c)\cr
\vec {\vphantom{b}a}&\approx \vec b\cr}$$
% The \vphantom is an invisible rule as tall as a `b'.
Le r\'esultat est d'ordre $O(n \log\log n)$.  D'o\`u
$$\sum_{i=1}^n x_i = x_1+x_2+\cdots+x_n
= {\rm Sum}(x_1,x_2,\ldots,x_n). \eqno(3)$$
et
$$dx\,dy = r\,dr\,d\theta\!.\eqno(4)$$
Le jeu de tous les $q$ tels que $q\le0$ s'\'ecrit~:
$$\{\,q\mid q\le0\, \}$$
Donc
$$\forall x\exists y\;P(x,y)\Rightarrow
\exists x\exists y\;P(x,y)$$
o\`u
$$P(x,y) \buildrel \rm def \over \equiv
\hbox{\rm tout pr\'edicat en $x$ et $y$} . $$
\bye
```

4 Concepts

Cette partie du livre contient les définitions et explications des concepts que nous utilisons pour décrire TEX. Les concepts comprennent les termes techniques que nous utilisons pour expliquer les commandes et des points importants qui ne sont vus nulle part ailleurs dans le livre

Les concepts sont classés dans l'ordre alphabétique. La page de garde intérieure contient une liste complète des concepts et les pages où ils sont expliqués. Nous vous suggérons de faire une copie de cette page intérieure et de la garder de coté pour que vous puissiez identifier et regarder un concept non familier immédiatement. Autant que possible, nous gardons notre terminologie cohérente avec celle de *The TEXbook*.

alignement. Un *alignement* est une construction pour aligner du matériel tels qu'un tableau, en colonne ou en rangée. Pour former un alignement vous devez (a) décrire le format des colonnes ou des rangées et (b) dire à TEX quel matériel va dans les colonnes ou les rangées. Un alignement tabulé ou un alignement horizontal est organisé comme une suite de rangées ; un alignement vertical est organisé comme une suite de colonnes. Nous décrirons d'abord les alignements tabulés et horizontaux et ensuite, plus brièvement, les alignements verticaux.

Les alignements tabulés sont définit par plain TEX. Ils sont plus simples mais moins flexibles que les alignements horizontaux. Ils diffèrent principalement sur la façon de décrire leurs formats.

Pour construire un alignement tabulé vous devez tout d'abord former une commande `\settabs` (p. 182) qui spécifie comment TEX doit diviser l'espace horizontal disponible en colonne. Ensuite vous procurez une suite de rangées à tabuler. Chaque rangée consiste en une séquence de contrôle `\+` (p. 181) suivi par une liste d'"entrées", c'est-à-dire des intersections rangées/colonnes. Des entrées qui se suivent sur une rangée sont séparées par une esperluette (`&`). La fin d'une rangée est indiquée par `\cr` après

sa dernière entrée. Si une rangée a moins d'entrées qu'il y a de colonnes dans l'alignement, TEX rempli jusqu'à la fin la rangée de blanc.

Tant qu'il est précédé d'une commande `\settabs`, vous pouvez mettre une rangée d'un alignement tabulé n'importe où dans votre document. En particulier, vous pouvez mettre autres choses entre les rangées d'alignement tabulé ou décrire plusieurs alignement tabulé avec un seul `\settabs`. Voici un exemple d'alignement tabulé :

```
{\hsize = 1.7 in \settabs 2 \columns
\+cattle&herd\cr
\+fish&school\cr
\+lions&pride\cr}
```

La commande `\settabs 2 \columns` de cet exemple (p. 182) demande à TEX de produire deux colonnes de largeurs égales. La longueur de la ligne est de 1.7 pouces. L'alignement composé ressemble à ceci :

cattle	herd
fish	school
lions	pride

Il y a une autre forme d'alignement tabulé dans lequel vous spécifiez les largeurs de colonne avec un exemple. La largeur de colonne de l'exemple détermine les largeurs de colonne du reste de l'alignement :

```
{\settabs\+cattle\quad&school\cr
\+cattle&herd\cr
\+fish&school\cr
\+lions&pride\cr}
```

Voici le résultat :

cattle	herd
fish	school
lions	pride

Les alignements horizontaux sont construits avec `\halign` (p. 184). TEX ajuste la largeur des colonnes d'un alignement horizontal en fonction du contenu des colonnes. Quand TEX rencontre la commande `\halign` qui débute un alignement horizontal, il examine d'abord toutes les rangées de l'alignement pour voir quelle est la largeur des entrées. Il fixe alors la largeur des colonnes pour accommoder les entrées les plus larges dans ces colonnes.

Un alignement horizontal géré par `\halign` est constitué d'un "préambule" qui indique le schéma des rangées suivi par les rangées elles-mêmes.

- Le préambule est constitué d'une suite de patrons, un pour chaque colonne. Le patron d'une colonne spécifie comment le texte de cette colonne doit être composé. Chaque patron doit inclure un seul caractère `#` pour indiquer où TEX doit substituer le texte d'une entrée dans le patron. Les patrons sont séparés par une esperluette (`&`) et la fin du préambule est indiquée par `\cr`. En procurant un patron approprié, vous pouvez obtenir des effets comme centrer une colonne, justifier

une colonne à gauche ou à droite ou composer une colonne dans une police particulière.

- Les rangées ont la même forme que dans l'alignement tabulé, sauf que vous ne mettez pas de `\+` au début de chaque rangée. Comme avant, des entrées sont séparées par `&` et la fin d'une rangée est indiquée par `\cr`. TEX traite chaque entrée comme un groupe, donc, chaque commande de changement de police ou autre assignement dans le patron de colonne ne prend effet que pour les entrées de cette colonne.

Le préambule et les rangées doivent être inclus entre les accolades qui suivent `\halign`. Chaque alignement `\halign` doit inclure son propre préambule.

Par exemple, l'alignement horizontal :

```
\tabskip=2pc
\halign{\hfil#\hfil &\hfil#\hfil &\hfil#\hfil \cr
  &&\it Table\cr
\noalign{\kern -2pt}
  \it Creature&\it Victual&\it Position\cr
\noalign{\kern 2pt}
  Alice&crumpet&left\cr
  Dormouse&muffin&middle\cr
  Hatter&tea&right\cr}
```

produit le résultat :

		Table
Creature	*Victual*	*Position*
Alice	crumpet	left
Dormouse	muffin	middle
Hatter	tea	right

Le `\tabskip` (p. 190) de cet exemple demande à TEX d'insérer `2pc` de ressort entre les colonnes. La commande `\noalign` (p. 189) d'insérer du matériel de mode vertical entre deux rangées. Dans cet exemple nous avons utilisé `\noalign` pour produire de l'espace supplémentaire entre les rangées de titre et les rangées de données, et aussi rapprocher "Table" et "Position". (Vous pouvez aussi utiliser `\noalign` avant la première rangée ou après la dernière.)

Vous pouvez aussi construire un alignement vertical avec la commande `\valign` (p. 185). Un alignement vertical est organisé comme une série de colonne plutôt que comme une série de rangée. Un alignement vertical suit les même règles qu'un alignement horizontal sauf que le rôle des rangées et des colonnes est inversé. Par exemple, l'alignement vertical :

```
{\hsize=0.6in \parindent=0pt
\valign{#\strut&#\strut&#\strut\cr
  one&two&three\cr
  four&five&six\cr
  seven&eight&nine\cr
  ten&eleven\cr}}
```

produit :

one	four	seven	ten
two	five	eight	eleven
three	six	nine	

La commande `\strut` (p. 173) dans le patron est nécessaire pour que les entrées de chaque rangée s'alignent proprement, c'est-à-dire, pour avoir une ligne de base commune et garder une distance uniforme entre les lignes de base.

Anatomie de TEX. *The TEXbook* décrit la façon dont TEX compile ses sources en termes de "tâches digestives" de TEX—ses "yeux", "bouche", "œsophage", "estomac" et "intestin". Savoir comment ces organes travaillent peut être utile quand vous essayer de comprendre de subtils aspects du raisonnement de TEX quand il digère un document.

- En utilisant ses "yeux", TEX lit les caractères du fichier source et les passe à sa bouche. Puisqu'un fichier source peut contenir des commandes `\input` (p. 255), TEX peut en effet détourner son regard d'un fichier à l'autre.
- En utilisant sa "bouche", TEX assemble les caractères en tokens et les passe à son œsophage. Chaque token est soit une séquence de contrôle ou un simple caractère. Une séquence contrôle commence toujours par un caractère d'échappement. Notez que les espaces et les fins de ligne sont des caractères de plein droit, de plus TEX compresse une suite d'espace en un seul token d'espace. Voir pages 46–47 de *The TEXbook* et 99–99 de la traduction française pour les règles avec lesquelles TEX assemble les caractères en tokens.
- En utilisant son "œsophage", TEX développe chaque macro, conditions, et constructions similaires qu'il trouve (voir pages 212–216 de *The TEXbook* et 999–999 de la traduction française) et passe la suite de tokens résultante à l'estomac de TEX. Développer un token peu faire apparaître d'autres tokens qui à leur tour ont besoin d'être développés. TEX parcoure ce développement de gauche à droite à moins que l'ordre soit modifié par une commande telle que `\expandafter` (p. 241). En d'autres mots, l'œsophage de TEX développe toujours le token le plus à gauche non-développé qu'il n'a pas encore envoyé vers l'estomac de TEX.
- En utilisant son "estomac", TEX exécute tous les groupes de token. Chaque groupe contient une commande primitive suivie par ses arguments, s'il y en a. La plupart des commandes sont de type "compose ce caractère", dont leur groupe n'est constitué que d'un token. Suivant les instructions données par les commandes, l'estomac de TEX assemble de plus en plus grandes unités, débutant par des caractères et finissant par des pages, et passe les pages à l'intestin de TEX. L'estomac de TEX exécute les taches de coupure de ligne—c'est-à-dire, couper chaque paragraphe en suite de lignes—et de coupure

de page—c'est-à-dire, couper une suite continues de lignes et autres matériels de mode vertical en pages.

- En utilisant ses "intestin", TEX transforme les pages produites par son estomac en une forme destinée à être lue par d'autres programmes. Il envoi alors la sortie transformée dans le fichier `.dvi`.

La plupart du temps, vous pouvez penser que les processus qui prennent place dans les yeux, bouches, œsophage, estomac et intestin de TEX ont lieu les uns après les autres. mais la vérité en la matière est que des commandes exécutées dans l'estomac de TEX peuvent influencer les étapes suivantes de la digestion. Par exemple, quand l'estomac de TEX rencontre la commande `\input` (p. 255), ses yeux commencent à lire un fichier différent ; quand l'estomac de TEX rencontre une commande `\catcode` (p. 259) spécifiant un code de catégorie pour le caractère *c*, l'interprétation de *c* par la bouche de TEX s'en trouve affecté. Et quand l'estomac de TEX rencontre une définition de macro, les développements chargés dans l'œsophage de TEX sont eux aussi affectés.

Vous pouvez comprendre comment les processus agissent les uns sur les autres en imaginant que chaque processus avale goulûment le rendu de son prédécesseur dès qu'il devient disponible. Par exemple, une fois que l'estomac de TEX a vu le dernier caractère du nom de fichier dans une commande `\input`, TEX fixe son regard immédiatement sur le premier caractère du fichier d'entrée indiqué.

argument. Un *argument* contient le texte passé à une commande. Les arguments d'une commande complètent la description de ce que la commande est supposée faire. La commande peut être soit une commande primitive, soit une macro.

Chaque commande primitive a sa propre convention sur la forme de ses arguments. Par exemple, la suite de tokens :

```
\hskip 3pc plus 1em
```

consiste en la commande '`\hskip`' et les arguments '`3pc plus 1em`'. Mais si vous écrivez :

```
\count11 3pc plus 1em
```

vous obtiendrez un effet entièrement différent. TEX traitera '`\count11`' comme une commande avec un argument '`3`', suivi par les tokens de texte ordinaires '`pc plus 1em`' (parce que les registres de compteur attendent l'affectation d'un nombre)—probablement pas ce dont vous aviez intention. L'effet de la commande, donc, sera d'assigner 3 au registre du compteur 11 (voir la discussion sur `\count`, p. 250).

Les macros, d'un autre coté, suivent toutes la même convention pour leurs arguments. Chaque argument passé à une macro correspond à un paramètre dans la définition de cette macro. Un paramètre de macro est soit "délimité", soit "non délimite". La définition de la macro détermine le nombre et la nature des paramètres de la macro et donc le nombre et la nature des arguments de la macro.

La différence entre un argument délimité et un argument non délimité tient à la façon qu'a TEX de décider où se termine l'argument.

- Un argument délimité est constitué des tokens à partir du début de l'argument jusqu'à, mais sans l'inclure, la suite de tokens particuliers qui servent comme délimiteur pour cet argument. Le délimiteur est spécifié dans la définition de la macro. Ainsi, vous ajoutez un argument délimité à une macro en écrivant l'argument lui-même, suivi par le délimiteur. Un argument délimité peut être vide, c'est-à-dire, ne pas avoir de texte du tout. Chaque accolade d'un argument délimité doit être appairé correctement, c'est-à-dire, chaque accolade gauche doit avoir une accolade droite correspondante et vice versa.
- Un argument non délimité consiste en un token simple ou une suite de tokens entouré d'accolades, comme ceci : '`{Here is {the} text.}`'. Malgré les apparences, les accolades extérieures ne forment pas un groupe—TEX ne les utilise que pour déterminer quel est l'argument. Chaque accolade interne, comme celles autour de '`the`', doivent être appairées correctement. Si vous faites une erreur et mettez trop d'accolades fermantes, TEX se plaindra d'un 'unexpected right brace' TEX se plaindra aussi si vous mettez trop d'accolades ouvrantes, mais vous obtiendrez *cette* plainte longtemps après l'endroit où vous pensiez finir l'argument (voir p. 283).

Voir "macro" (p. 73) pour plus d'information sur les paramètres et les arguments. Vous trouverez les règles précises concernant les arguments délimités et non délimités dans pages 203–204 de *The TEXbook* et 999–999 de la traduction française.

ASCII. ASCII est l'abréviation de "American Standard Code for Information Interchange". Il y a 256 caractères ASCII, chacun avec son propre numéro de code, mais seuls les 128 premiers ont été standardisés. Vous pouvez trouver leurs significations dans une "table de codes" ASCII comme celle de la page 367 de *The TEXbook* et 999 de la traduction française. Les caractères 32–126 sont des "caractères imprimables", comme les lettres, les chiffres et les signes de ponctuation. les caractères restants sont des "caractères de contrôle" qui sont utilisés spécialement (dans l'industrie informatique, pas dans TEX) pour contrôler les entrées/sorties et les instruments de communication de données. Par exemple, le code ASCII 84 correspond à la lettre 'T', tandis que le code ASCII 12 correspond à la fonction "form feed" (interprété par la plupart des imprimantes par "commence une nouvelle page"). Bien que le standard ASCII spécifie des significations pour les caractères de contrôle, beaucoup de fabricants d'équipements tels que modems et imprimantes ont utilisé les caractères de contrôle pour des usages autres que le standard.

La signification d'un caractère dans TEX est normalement liée à sa signification dans le standard ASCII, et les polices qui contiennent des caractères imprimables ASCII ont normalement ces caractères aux mêmes positions que leurs contreparties ASCII. Mais certaines polices, notam-

ment celles utilisées en mathématique, remplacent les caractères imprimables ASCII par d'autres caractères sans aucune relation avec les caractères ASCII. Par exemple, La police Computer Modern mathématique `cmsy10` a le symbole mathématique '∀' à la place du chiffre ASCII '8'.

assignement. Un *assignement* est une construction qui dit à TEX d'assigner une valeur à un registre, à un de ces paramètres internes, à une entré dans une de ces tables internes ou à une séquence de contrôle. quelques exemples d'assignements sont :

```
\tolerance = 2000
\advance\count12 by 17
\lineskip = 4pt plus 2pt
\everycr = {\hskip 3pt \relax}
\catcode\‘@ = 11
\let\graf = \par
\font\myfont = cmbx12
```

Le premier assignement indique que TEX doit assigner la valeur numérique `2000` au paramètre numérique `\tolerance`, c'est-à-dire, met la valeur de `\tolerance` à 2000. Les autres assignements sont similaires. Le '`=`' et les espaces sont optionnels, donc vous pouvez aussi écrire le premier assignement plus brièvement :

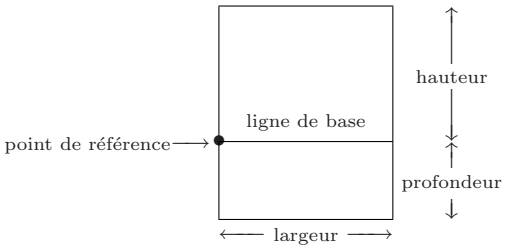
```
\tolerance2000
```

Voir pages 276–277 de *The TEXbook* et 999–999 de la traduction française pour la syntaxe détaillée des assignements.

boîtes. Une *boîte* est un rectangle de matière à composer. Un simple caractère est une boîte par lui-même, et une page entière est aussi une boîte. TEX forme une page comme une suite de boîtes contenant des boîtes contenant des boîtes. La boîte supérieure est la page elle-même, les boîtes inférieures sont souvent de simples caractères et de simples lignes sont des boîtes qui sont quelque part au milieu.

TEX effectue la plupart de ses activités de construction de boîte implicitement quand il construit des paragraphes et des pages. Vous pouvez construire des boîtes explicitement en utilisant nombre de commandes de TEX, notamment `\hbox` (p. 166), `\vbox` (p. 167) et `\vtop` (p. 167). La commande `\hbox` construit une boîte en assemblant des plus petites boîtes horizontalement de gauche à droite ; il opère sur une liste horizontale et donne une hbox (boîte horizontale). les commandes `\vbox` et `\vtop` construisent une boîte en assemblant de plus petites boîtes verticalement de haut en bas ; elles opèrent sur une liste verticale et donne une vbox (boîte verticale). ces listes horizontales et verticales peuvent ne pas inclure que des boîtes plus petites mais plusieurs autres sortes d'entités, par exemple, un ressort et des crénages.

Une boîte a une hauteur, une profondeur et une largeur, comme ceci :



La ligne de base est comme une des lignes-guide claires d'un bloc de papier ligné. Les boîtes de lettres telles qu'un 'g' s'étendent sous la ligne de base ; Les boîtes de lettres telles qu'un 'h' ne le font pas. La hauteur d'une boîte est la distance par laquelle une boîte s'étend au-dessus de sa ligne de base, tandis que sa profondeur est la distance par laquelle elle s'étend sous sa ligne de base. Le point de référence d'une boîte est l'endroit où sa ligne de base rejoint son coté gauche.

TEX construit une hbox H d'une liste horizontale en désignant un point de référence pour H et en enfilant les items dans la liste H un par un de gauche à droite. chaque boîte de la liste est placée de telle façon que sa ligne de base coïncide avec la ligne de base de H, c'est-à-dire que les boîtes la composant sont alignées horizontalement[1]. La hauteur de H est la hauteur de la boîte la plus grande de la liste, et la profondeur de H est la profondeur de la boîte la plus profonde de la liste. La largeur de H est la somme des largeurs de tous les items dans la liste. Si quelques-uns de ces items sont des ressorts et que TEX a besoin de rétrécir ou d'étirer le ressort, la largeur de H sera plus large ou plus étroite en conséquence. Voir la page 77 de *The TEXbook* et 99 de la traduction française pour les détails.

De même, TEX construit une vbox V d'une liste verticale en désignant un point de référence temporaire pour V et ensuite en enfilant les items dans la liste V un par un de haut en bas. Chaque boîte dans la liste est placée de telle façon que leur point de référence est aligné verticalement avec le point de référence de V[2]. Quand chaque boîte autre que la première est ajoutée à V, TEX met un ressort interligne juste au-dessus d'elle. (Ce ressort interligne n'a pas d'équivalent pour les hbox.) La largeur de V est la largeur de la boîte la plus large de la liste, et les extensions verticales (hauteur plus profondeur) de V est la somme des éléments verticaux étendus de tous les items de la liste.

La différence entre `\vbox` et `\vtop` est sur leur façon de partitionner l'extension verticale de V en hauteur et en largeur. Le choix du point de référence de V détermine cette partition.

---

[1] Si une boîte est haussée ou baissée avec `\raise` ou `\lower`, TEX utilise son point de référence avant le déplacement quand il la place.

[2] Si une boîte est déplacée à gauche ou à droite avec `\moveleft` ou `\moveright`, TEX utilise son point de référence avant le déplacement quand il la place.

- Pour `\vbox`, TEX place le point de référence sur une ligne horizontale avec le point de référence du dernier composant boîte ou trait de V, sauf que si la dernière boîte (ou le dernier trait) est suivie d'un ressort ou d'un crénage, TEX place le point de référence tout en bas de V[3].
- Pour `\vtop`, TEX place le point de référence sur une ligne horizontale avec le point de référence du premier composant boîte ou trait de V, sauf que si la première boîte (ou le premier trait) est précédée par un ressort ou un crénage, TEX place le point de référence tout en haut de V.

A proprement parler, alors, `\vbox` met le point de référence près du haut du vbox et `\vtop` le met près du haut. Quand vous voulez aligner une rangée de vbox pour que leur sommet s'aligne horizontalement, vous utiliserez normalement `\vtop` plutôt que `\vbox`. Voir les pages 78 et 80–81 de *The TEXbook* et 99 et 99–99 de la traduction française pour les détails sur comment TEX construit des vbox.

Vous avez une certaine liberté dans la construction de boîtes. Le matériel à composer dans une boîte peut s'étendre au-delà des bords de la boîte comme il le fait pour certaines lettres (pour la plupart italiques ou penchées). Les composants boîtes des grandes boîtes peuvent se chevaucher. Une boîte peut avoir une largeur, une profondeur ou une hauteur négative, des boîtes comme celle-ci ne sont pas souvent utiles.

Vous pouvez sauvegarder une boîte dans un registre de boîte et la rappeler plus tard. Avant d'utiliser un registre de boîte, Vous devez la réserver et lui donner un nom avec la commande `\newbox` (p. 252). Voir "registre" (p. 90) pour plus d'information sur les registres de boîtes.

caractère. TEX travaille avec des *caractères* provenant de deux contextes : des caractères du source, quand il lit et des caractères de sortie, quand il compose. TEX transforme la plupart des caractères saisis dans le caractère de sortie qui lui correspond. Par exemple, il transforme normalement la lettre saisie '`h`' en 'h' composé dans la police courante. Ce n'est pas le cas, néanmoins, pour un caractère saisi tel que '`$`' qui a une signification spéciale.

TEX obtient ses caractères saisis en les lisant à partir fichier source (ou de votre terminal) et en développement des macros. Ce sont les seuls moyens par lesquels TEX puisse acquérir un caractère saisi. Chaque caractère d'entrée a un numéro de code correspondant à sa position dans la table de code ASCII. Par exemple, la lettre '`T`' a le code ASCII 84.

Quand TEX lit un caractère, il lui attache un code de catégorie. Le code de catégorie affecte la façon dont TEX interprète le caractère une fois qu'il a été lu. TEX détermine (et se souvient) des codes de catégorie des caractères d'une macro quand il lit sa définition. Comme TEX lit les caractères avec ses yeux (voir "Anatomie de TEX", p. 48) il fait quelques "filtrages" comme condenser des suites de caractères espace en un espace

[3] La profondeur est limitée par le paramètre `\boxmaxdepth` (p. 169).

seul. Voir les pages 46–48 de *The TEXbook* et 99–99 de la traduction française pour les détails de ce filtrage.

Les "caractères de contrôle" ASCII ont les codes 0–31 et 127–255. Ils ne sont pas visualisables ou provoque des choses étranges si vous essayez de les afficher. Néanmoins, on en a parfois besoin dans les sources TEX, donc TEX a une façon spéciale de les noter. Si vous saisissez '`^^`c', où c est n'importe quel caractère, vous obtenez le caractère dont le code ASCII est soit plus grand soit plus petit de 64 que le code ASCII de c. La plus grande valeur de code acceptable en utilisant cette notation est 127 donc la notation n'est pas ambiguë. Trois instances particulièrement communes de cette notation sont '`^^M`' (le caractère ASCII ⟨retour⟩), '`^^J`' (le caractère ASCII ⟨saut de page⟩) et '`^^I`' (le caractère ASCII ⟨tabulation horizontale⟩).

TEX a aussi une autre notation pour indiquer la valeur des codes ASCII qui fonctionne pour tous les codes de caractère de 0 à 255. Si vous saisissez '`^^`xy', où x et y sont des chiffres hexadécimaux '`0123456789abcdef`', vous obtenez le caractère dont le code est spécifié.(Des lettres minuscules sont nécessaires ici.) TEX opte pour l'interprétation des "chiffres hexadécimaux" quand il a le choix, donc vous ne devez pas faire suivre un caractère comme '`^^a`' par un chiffre hexadécimal minuscule—si vous le faites, vous aurez la mauvaise interprétation. Si vous avez besoin d'utiliser cette notation, vous trouverez pratique d'avoir une table des codes ASCII.

Un caractère de sortie est un caractère de composition. Une commande qui produit un caractère de sortie signifie "Produit une boîte contenant un caractère numéro n dans la police courante", où n est déterminé par la commande. TEX produit votre document composé en combinant de telles boîtes avec d'autres éléments typographiques et les arrangent sur la page.

Un caractère d'entrée dont le code de catégorie est 11 (lettre) ou 12 (autre) agit comme une commande pour produire le caractère de sortie correspondant. En plus, vous pouvez demander à TEX de produire un caractère n en entrant la commande '`\char` n' (p. 105), où n est un nombre entre 0 et 255. Les commandes '`h`', `\char`'`h` et `\char104` ont toutes le même effet. (104 est le code ASCII de 'h'.)

caractère actif. Un *caractère actif* est un caractère possédant une définition, c'est-à-dire, une définition de macro, qui lui est associée. Vous pouvez penser à un caractère actif comme à une séquence de contrôle d'un type spécial. Quand TEX rencontre un caractère actif, il exécute la définition associée avec le caractère. Si TEX rencontre un caractère actif qui n'a pas de définition associée, il se plaindra d'un "undefined control sequence".

Un caractère actif a un code de catégorie à 13 (la valeur d `\active`). Pour définir un caractère actif, vous devez d'abord utiliser la commande `\catcode` (p. 259) pour le rendre actif et ensuite donner la définition du caractère, en utilisant une commande comme `\def`, `\let` ou `\chardef`. La définition d'un caractère actif à la même forme que la définition d'une séquence de contrôle. Si vous essayez de définir la macro pour un caractère

actif avant d'avoir rendu le caractère actif, TEX se plaindra d'un "missing control sequence".

Par exemple, le caractère tilde (~) est défini comme un caractère actif dans plain TEX. Il produit un espace entre deux mots mais relie ces mots pour que TEX ne transforme pas cet espace en coupure de ligne. Plain TEX définit '~' par les commandes :

```
\catcode `~ = \active \def~{\penalty10000\␣}
```

(`\penalty` inhibe une coupure de ligne et '`\␣`' insère un espace.)

caractère d'échappement. Un *caractère d'échappement* introduit une séquence de contrôle. Le caractère d'échappement dans plain TEX est l'antislash (\). Vous pouvez changer le caractère d'échappement de c_1 vers c_2 en réassignant les codes de catégorie de c_1 et c_2 avec la commande `\catcode` (p. 259). Vous pouvez aussi définir des caractères d'échappement additionnels de la même façon. Si vous voulez composer du matériel contenant des caractères d'échappement littéraux, vous devez soit (a) définir une séquence de contrôle qui transforme le caractère d'échappement imprimé soit (b) changer temporairement son code de catégorie, en utilisant la méthode montrée en page 2. La définition :

```
\def\\{$\backslash$}
```

est un moyen de créer une séquence de contrôle qui le transforme en '\' (un antislash composé dans une police mathématique).

Vous pouvez utiliser le paramètre `\escapechar` (p. 234) pour spécifier comment le caractère de contrôle est représenté dans des séquences de contrôle synthétisées, c'est-à-dire, celles créées par `\string` et `\message`.

césure. TEX césure les mots automatiquement en compilant votre document. TEX n'est pas un acharné de l'insertion de césure, préférant trouver à la place une bonne coupure de ligne en ajustant l'espace entre les mots et en déplaçant les mots d'une ligne à l'autre. TEX est assez intelligent pour comprendre les césures qui sont déjà dans les mots

Vous pouvez contrôler les césures de TEX de plusieurs façons :

- Vous pouvez dire à TEX d'autoriser une césure à un endroit particulier en insérant une césure optionnelle avec la commande `\-` (p. 132).
- Vous pouvez dire à TEX comment couper des mots particuliers dans votre document avec la commande `\hyphenation` (p. 133).
- Vous pouvez englober un mot dans une hbox pour empêcher TEX de le couper.
- Vous pouvez fixer la valeur des pénalités comme avec `\hyphenpenalty` (p. 131).

Si un mot contient une césure explicite ou optionnelle, TEX ne le coupera jamais ailleurs.

classe. La *classe* d'un caractère spécifie le rôle de ce caractère dans les formules mathématiques. La classe d'un caractère est encodée dans son mathcode. Par exemple, le signe égal '`=`' est dans la classe 3 (Relation). TEX utilise sa connaissance de classes de caractère pour décider combien d'espace mettre entre différents composants d'une formule mathématique. Par exemple, voici une formule mathématique affichée comme TEX l'imprime normalement et avec la classe de chaque caractère modifié aléatoirement :

$$a + (b - a) = a \qquad a+ (b- a)=a$$

Voir page 226 de ce livre pour une liste des classes et la page 154 de *The TEXbook* et 999 de la traduction française pour leurs significations.

codes de catégorie. Le *code de catégorie* d'un caractère détermine le rôle de ce caractère dans TEX. Par exemple, TEX assigne un certain rôle aux lettres, un autre au caractère espace, et ainsi de suite. TEX attache un code à chaque caractère qu'il lit. Quand TEX lit la lettre '`r`', par exemple, il lui attache normalement le code de catégorie 11 (lettre). Pour de si simple applications TEX vous n'avez pas à vous préoccuper des codes de catégorie, mais ils deviennent importants quand vous essayez de terminer des effets spéciaux.

Les codes de catégorie ne s'appliquent qu'aux caractères que TEX lit dans les fichiers source. Une fois qu'un caractère a passé l'œsophage de TEX (voir "Anatomie de TEX", p. 48) et a été interprété, son code de catégorie ne sert plus. Un caractère que vous produisez avec la commande `\char` (p. 105) n'a pas de code de catégorie parce que `\char` est une instruction pour que TEX produise un certain caractère d'une certaine police. Par exemple, le code ASCII pour '\' (le caractère d'échappement usuel) est 92. Si vous saisissez '`\char92 grok`', Ce n'est *pas* équivalent à `\grok`. à la place il demande à TEX de composer '*c*grok', où *c* est le caractère en position 92 de la table de code de la police courante.

Vous pouvez utiliser la commande `\catcode` (p. 259) pour réassigner le code de catégorie de tout caractère. En changeant les code de catégorie vous pouvez changer les rôles de caractères variés. Par exemple, si vous saisissez '`\catcode`‘`\@ = 11`', le code de catégorie du signe arrose (`@`) sera celui d'une "lettre". Vous pourrez alors avoir '`@`' dans le nom d'une séquence de contrôle.

Voici une liste des codes de catégorie tels qu'ils sont définis dans plain TEX (voir p. 54 pour une explication de la notation `^^`), ainsi que les caractères de chaque catégorie :

Code	*Signification*	
0	Caractère d'échappement	`\`
1	Début d'un groupe	`{`
2	Fin d'un groupe	`}`
3	basculement en mathématique	`$`
4	Alignement tab	`&`

5 Fin d'une ligne `^^M` ≡ ASCII ⟨retourchariot⟩
6 Paramètre de macro `#`
7 Exposant `^` et `^^K`
8 Indice `_` et `^^A`
9 caractère ignoré `^^@` ≡ ASCII ⟨nul⟩
10 Espace ␣ et `^^I` ≡ ASCII ⟨tabulation⟩
11 Lettre `A` ... `Z` et `a` ... `z`
12 Autres caractères (tout ce qui n'est pas ci-dessus ou ci-dessous)
13 Caractère actif `~` et `^^L` ≡ ASCII ⟨sautdepage⟩
14 Caractère de commentaire `%`
15 Caractère invalide `^^?` ≡ ASCII ⟨suppression⟩

Mise à par ceux des catégories 11–13, tous les caractères d'une catégorie particulière produisent le même effet. Par exemple, supposez que vous saisissiez :

```
\catcode`\[ = 1 \catcode`\] = 2
```

Alors les caractères crochets droits et gauches deviennent équivalent aux caractères de début et de fin d'un groupe, les caractères accolades droites et gauches. Avec ces définitions '`[a b]`' est un groupe valide, ainsi que '`[a b}`' et '`{a b]`'.

Les caractères des catégories 11 (lettre) et 12 (autres caractères) agissent comme des commandes ce qui signifie "produit une boîte contenant ce caractère composé dans la police courante". La seule distinction entre les lettres et les "autres" caractères est que les lettres peuvent apparaître dans des mots de contrôle mais pas les "autres" caractères.

Un caractère de catégorie 13 (actif) agit comme une séquence de contrôle à lui tout seul. TEX se plaint s'il rencontre un caractère actif qui n'est pas associé avec une définition.

Si TEX rencontre un caractère invalide (catégorie 15) dans votre source, il s'en plaindra.

Les caractères '`^^K`' et '`^^A`' ont été inclus dans les catégories 8 (indice) et 9 (exposant), même si ces significations ne suivent pas l'interprétation du standard ASCII. C'est parce que certains claviers, tout du moins certain à l'université de Stanford d'où TEX est originaire, n'ont pas de touche flèche vers le bas et flèche vers le haut qui génèrent ces caractères.

Il y a une subtilité sur la façon dont TEX assigne les codes de catégories qui peut vous déboussoler si vous ne la connaissez pas. TEX a parfois besoin de rechercher un caractère deux fois quand il fait un survol initial : la première fois pour la fin d'une précédente construction, c'est-à-dire, une séquence de contrôle et ensuite pour transformer ce caractère en token. TEX n'a pas besoin d'assigner le code de catégorie avant d'avoir vu le *second* caractère. Par exemple :

```
\def\foo{\catcode`\$ = 11 }% Make $ be a letter.
\foo$ % Produces a `$'.
\foo$ % Undefined control sequence `foo$'.
```

Ce bout de code TEX produit '$' dans la sortie. Quand TEX voit le '`$`' sur la seconde ligne, il recherche la fin du nom d'une séquence de contrôle. Puisque le '`$`' n'est pas encore une lettre, il marque la fin de '`\foo`'. Ensuite, TEX développe la macro '`\foo`' et change le code de catégorie de '`$`' en 11 (lettre). Alors, TEX lit le '`$`' "pour de vrai". Puisque '`$`' est maintenant une lettre, TEX produit une boîte contenant le caractère '`$`' de la police courante. Quand TEX voit la troisième ligne, il traite '`$`' comme une lettre et donc considère qu'elle fait partie du nom de la séquence de contrôle. Avec comme résultat qu'il se plaint d'un "undefined control sequence `\foo$`".

TEX se comporte de cette façon même lorsque le caractère de terminaison est une fin de ligne. Par exemple, supposez que la macro `\fum` active le caractère de fin de ligne. Alors si `\fum` apparaît sur une ligne ℓ elle-même, TEX interprétera d'abord la fin de la ligne ℓ comme la fin de la séquence de contrôle `\fum` et ensuite *réinterprétera* la fin de la ligne de ℓ comme un caractère actif.

commande. Une *commande* demande à TEX de faire une certaine action. Chaque token qui atteint l'estomac de TEX(voir "Anatomie de TEX", p. 48) agit comme une commande, sauf celles qui font partie des arguments d'une autre commande (voir dessous). Une commande peut être appelée par une séquence de contrôle, par un caractère actif ou par un caractère ordinaire. Il peut sembler bizarre que TEX traite un caractère ordinaire comme une commande, mais en fait c'est ce qu'il fait : Quand TEX voit un caractère ordinaire il construit une boîte contenant ce caractère composé dans la police courante.

Une commande peut avoir des arguments. Les arguments d'une commande sont de simples tokens ou groupes de tokens qui complètent la description de ce que la commande est supposée faire. Par exemple, la commande '`\vskip 1in`' demande à TEX de sauter 1 pouce verticalement. Elle a un argument '`1in`', qui est constitué de trois tokens. La description de ce que `\vskip` est supposée faire serait incomplète sans spécifier de combien elle est supposée sauter. Les tokens dans les arguments d'une commande ne sont pas considérés eux-mêmes comme des commandes.

Quelques exemples de différentes sortes de commande de TEX sont :

- Des caractères ordinaires, comme '`W`', qui demande à TEX de produire une boîte contenant un 'W' composé.
- Des commandes de modification de police, telles que `\bf`, qui commence une composition en gras
- Des accents, tels que `\‘`, qui produit un accent grave comme dans 'è'
- Des symboles spéciaux et des ligatures, comme `\P` (¶) et `\ae` (æ)
- Des paramètres, tels que `\parskip`, le montant du ressort que TEX mets entre les paragraphes
- des symboles mathématiques, tels que `\alpha` (α) et `\in` ($\in$)
- Des opérateurs mathématiques, tels que `\over`, qui produit une fraction

constantes décimales. See "nombre" (p. 83).

construction de page. Voir "page" (p. 85).

coupure de ligne. Une *coupure de ligne* est un endroit dans votre document où TEX termine une ligne quand il compose un paragraphe. Quand TEX compile votre document, il collecte le contenu de chaque paragraphe dans une liste horizontale. Quand il a collecté un paragraphe entier, il analyse la liste pour trouver ce qu'il considère comme étant les meilleures coupures de ligne possibles. TEX associe des "démérites" avec différents symptômes de coupure de ligne non attractive—des lignes ayant trop ou trop peu d'espace entre les mots, des lignes consécutives se terminant par des césures, et ainsi de suite. Il choisit alors les coupures de lignes de manière à minimiser le nombre total de démérites. Voir les pages 96–101 de *The TEXbook* et 99–999 de la traduction française pour une description complète des règles de coupure de ligne de TEX.

Vous pouvez contrôler le choix des coupures de ligne de TEX de plusieurs manières :

- vous pouvez insérer une pénalité (p. 127) quelque part dans la liste horizontale que TEX construit quand il forme un paragraphe. Une pénalité positive décourage TEX de couper la ligne à cet endroit, tandis qu'une pénalité négative—un bonus, en quelque sorte—encourage TEX à couper la line à cet endroit. Une pénalité de 10000 ou plus empêche une coupure de ligne, tandis qu'une pénalité de −10000 ou moins force une coupure de ligne. Vous pouvez obtenir les mêmes effets avec les commandes `\break` et `\nobreak` (pp. 126, 127).
- Vous pouvez dire à TEX d'autoriser une césure à un endroit particulier en insérant une césure optionnelle avec la commande `\-` (p. 132), ou sinon en contrôlant comment TEX met des césures dans votre document (voir "césure", p. 55).
- Vous pouvez dire à TEX d'autoriser une coupure de ligne après un (/) slash entre deux mots en insérant un `\slash` (p. 128) entre eux, par exemple, '`furlongs\slash fortnight`'.
- Vous pouvez dire à TEX de ne pas couper une ligne entre deux mots particuliers en insérant une punaise (`~`) entre ces mots.
- Vous pouvez ajuster les pénalités associées avec les coupures de ligne en assignant des valeurs différentes aux paramètres de coupures de lignes de TEX.
- Vous pouvez englober un mot ou une séquence de mots dans une hbox, empêchant ainsi TEX de couper la ligne où que cela soit dans l'hbox.

Il est pratique de savoir où TEX peut couper une ligne :

- sur un ressort, sous réserve que :
  1) l'objet précédant le ressort est un des suivants : une boîte, un objet optionnel (par exemple, une césure optionnelle), la fin

d'une formule mathématique, un élément extraordinaire, ou du matériel vertical produit par `\mark`, `\vadjust` ou `\insert`

2) Le ressort ne fait pas partie d'une formule mathématique

Quand TEX coupe une ligne sur un ressort, il fait la coupure du coté gauche de l'espace du ressort et oublie le reste du ressort.

- sur un crénage qui est immédiatement suivi par un ressort, à condition que ce crénage ne soit pas dans une formule mathématique
- à la fin d'une formule mathématique qui est immédiatement suivie par un ressort
- sur une pénalité, même dans une formule mathématique
- sur une césure optionnelle

Quand TEX coupe une ligne, il efface toute séquence de ressort, crénage et pénalité qui suivent le point de coupure. Si une telle séquence est suivie par le début d'une formule mathématique, il efface aussi tout crénage produit au début de la formule.

coupure de page. Une *coupure de page* est un endroit dans votre document où TEX termine une page et (sauf à la fin du document) en commence une nouvelle. Voir "page" (p. 85) pour le processus que TEX traverse pour choisir une coupure de page.

Vous pouvez contrôler le choix de coupure de page de TEX de plusieurs façon :

- Vous pouvez insérer une pénalité (p. 142) entre deux éléments de la liste verticale principale. Une pénalité positive décourage TEX de couper la page à cet endroit, tandis qu'une pénalité négative—un bonus, autrement dit—encourage TEX de couper la page à cet endroit. Une pénalité de 10000 ou plus prévient une coupure de page, tandis qu'une pénalité de −10000 ou moins force une coupure de page. Vous pouvez obtenir le même effet avec les commandes `\break` et `\nobreak` (p. 142).
- Vous pouvez ajuster les pénalités associées aux coupures de page en assignant différentes valeurs aux paramètres de coupure de page de TEX.
- Vous pouvez englober une suite de paragraphes ou autres éléments dans la liste verticale principale avec une vbox, ceci empêche TEX de couper la page n'importe où dans la suite.

Une fois que TEX a choisi une coupure de page, il place la portion de la liste verticale principale qui précède la coupure en `\box255`. Il appelle alors la routine de sortie courante pour exécute `\box255` et éventuellement envoyer son contenu dans le fichier `.dvi`. La routine de sortie doit aussi inclure des insertions, telles que des notes de pied de page, que TEX a accumulé en exécutant la page.

Il est utile de connaître les endroits où TEX peut couper une page :

- Sur le ressort, à condition que l'élément précédent le ressort soit une boîte, un élément extraordinaire, une marque ou une insertion.

Quand TEX coupe une page sur un ressort, il fait la coupure devant l'espace du ressort et oublie le reste du ressort.

- Sur un crénage qui est immédiatement suivi par un ressort.
- Sur une pénalité, vraisemblablement entre les lignes d'un paragraphe.

Quand TEX coupe une page, il efface toute séquence de ressort, crénages et éléments de pénalité qui suivent le point de coupure.

crénage. Un *crénage* indique un changement de l'espacement normal entre les éléments d'une liste horizontale ou verticale. Un crénage peut être soit positif, soit négatif. En mettant un crénage positif entre deux éléments, vous les éloignez du montant du crénage. en mettant un crénage positif entre deux éléments, vous les rapprochez du montant du crénage. Par exemple, ce texte :

```
11\quad 1\kern1pt 1\quad 1\kern-.75pt 1
```

produit des paires de lettres qui ressemblent à ceci :

11 11 11

Vous pouvez utiliser des crénages en mode vertical pour ajuster l'espace entre des paires de lignes particulières.

Un crénage de taille *d* est très similaire à un élément ressort qui a une taille *d* sans étirement ni rétrécissement. Le crénage et le ressort insèrent ou enlèvent de l'espace entre deux éléments voisins. La différence essentielle est que TEX considère deux boîtes n'ayant qu'un crénage entre elles comme étant liées. Cela fait que, TEX ne coupe pas une ligne ou une page sur un crénage sauf si le crénage est immédiatement suivi d'un ressort. garder cette différence à l'esprit quand vous choisissez d'utiliser un crénage ou un élément ressort pour un usage particulier.

TEX insère automatiquement des crénages entre des paires particulières de lettres adjacentes, ajustant ainsi l'espace entre ces lettres et mettant en valeur l'apparence de votre document composé. Par exemple, la police Computer Modern romaine de 10 points contient un crénage pour la paire 'To' qui remet le coté gauche du 'o' sous le 'T'. Sans le crénage, vous obtiendriez "Top" au lieu de "Top"— la différence est légère mais visible. Le fichier de métrique (fichier `.tfm`) de chaque police spécifie le placement et la taille des crénages que TEX insère automatiquement quand il met du texte dans cette police.

dimension. Une *dimension* spécifie une distance, c'est, une mesure linéaire d'espace. Vous utilisez des dimensions pour spécifier des tailles de choses, comme les grandeurs d'une ligne. Les imprimeurs dans les pays de langue Anglaise mesurent traditionnellement les distances en points et en picas, tandis que les imprimeurs d'Europe continentale mesurent traditionnellement les distances en points Didot et en ciceros. Vous pouvez utiliser ces mesures ou d'autres, comme les pouces, qui peuvent vous être

plus familier. Les unités de mesure indépendante des polices que TEX comprend sont :

`pt`	point (72.27 points = 1 pouce)
`pc`	pica (1 pica = 12 points)
`bp`	big point (72 gros points = 1 pouce)
`in`	pouce
`cm`	centimètre (2.54 centimètres = 1 pouce)
`mm`	millimètre (10 millimètres = 1 centimètre)
`dd`	point didôt (1157 point didôts = 1238 points)
`cc`	cicéro (1 cicero = 12 points Didot)
`sp`	point d'échelle (65536 points d'échelle = 1 point)

Deux unités de mesure supplémentaire sont associées avec chaque police : '`ex`', une mesure verticale correspondant habituellement à la hauteur de la lettre 'x' de la police et '`em`', une mesure horizontale habituellement égale à la taille du point de la police et environ la largeur de la lettre 'M' de la police. Finalement, TEX procure trois unité de mesure "infinies" : '`fil`', '`fill`' et '`filll`', dans l'ordre de force croissant.

Une dimension est écrite comme un facteur, c'est-à-dire, un multiplieur, suivi par une unité de mesure. Le facteur peut être soit un nombre entier, soit une constante décimale contenant un point décimal ou une virgule décimale. Le facteur peut être précédé par un signe plus ou moins, donc une dimension peut être positive ou négative. L'unité de mesure doit être là, même si le nombre est zéro. Des espaces entre le nombre et l'unité de mesure sont permis mais facultatifs. Vous trouverez une définition précise d'une dimension sur la page 270 de *The TEXbook* et 999 de la traduction française. Voici quelques exemples de dimensions :

```
5.9in    0pt    -2,5 pc    2fil
```

La dernière représente un ordre de distance infinie.

Une distance infinie écrase toute distance finie ou toute distance infinie plus légère. Si vous additionnez `10in` à `.001fil`, vous obtiendrez `.001fil` ; si vous additionnez `2fil` à `-1fill` vous aurez `-1fill` ; et ainsi de suite. TEX n'accepte des distances infinies que quand vous avez spécifié l'étirement et le rétrécissement du ressort.

TEX multiplie toutes les dimensions de votre document par un facteur de magnification $f/1000$, où f est la valeur du paramètre `\mag`. Puisque la valeur par défaut de `\mag` est 1000, le cas normal est que votre document est composé comme spécifié. Vous pouvez spécifier une dimension comme elle sera mesurée dans le document final indépendamment d'une magnification en mettant '`true`' devant l'unité. Par exemple, '`\kern 8 true pt`' produit un crénage de 8 points quelle que soit la magnification.

disposition de page. Quand vous concevez un document, vous devez décider de son *ajustement de page* : la taille de la page, les marges des quatre cotés, les entêtes et pieds de page, s'il y en a, qui apparaissent en haut et en bas de la page et la taille de l'espace entre le corps du texte et

l'entête ou le pied de page. TEX en défini par défaut. Il définit une page de 8 1/2 par 11 pouces avec des marges d'approximativement un pouce de chaque coté, sans entête et avec un pied de page constitué d'un numéro de page centré.

Les marges sont déterminées conjointement par les quatre paramètres `\hoffset`, `\voffset`, `\hsize` et `\vsize` (voir "marges", page 77, pour des conseils sur la façon de les ajuster). L'entête consiste normalement en une simple ligne qui apparaît en haut de chaque page, dans l'aire de la marge du haut. Vous pouvez la définir en assignant une liste de token dans le paramètre `\headline` (p. 149). Similairement, le pied de page consiste normalement en une simple ligne qui apparaît en bas de chaque page, dans l'aire de la marge du bas. Vous pouvez la définir en assignant une liste de token dans le paramètre `\footline` (p. 149). Par exemple, l'entrée :

```
\headline = {Baby's First Document\dotfill Page\folio}
\footline = {\hfil}
```

produit une ligne d'entête comme celle-ci sur chaque page :

Baby's First Document . . . . . Page 19

et aucune ligne de pied de page.

Vous pouvez utiliser des marques pour placer le sujet d'une section de texte courante dans l'entête ou le pied de page. Voir "marque" (p. 77) pour une explication sur comment faire cela.

délimiteur. Un *délimiteur* est un caractère mis comme frontière visible d'une formule mathématique. La propriété essentielle d'un délimiteur est que TEX peut ajuster sa taille par rapport à la taille verticale (hauteur plus profondeur) de la sous-formule. De plus, TEX n'exécute d'ajustement que si le délimiteur apparaît dans un "contexte de délimiteur", plus précisément, comme argument d'une des commandes `\left`, `\right`, `\overwithdelims`, `\atopwithdelims`, ou `\abovewithdelims` (voir les pp. 209, 212). Les contextes de délimiteur incluent aussi tout argument à une macro qui utilise l'argument dans un contexte de délimiteur.

Par exemple, les parenthèses gauches et droites sont des délimiteurs. Si vous utilisez des parenthèses dans un contexte de délimiteur autour d'une formule, TEX rend les parenthèses assez grandes pour entourer la boîte qui contient la formule (à condition que les polices que vous utilisez aient des parenthèses assez grandes). Par exemple :

```
$$ \left( a \over b \right) $$
```

donne :

$$\left(\frac{a}{b}\right)$$

Ici TEX a rendu les parenthèses assez grandes pour s'accorder à la fraction. Mais si vous écrivez, à la place :

```
$$({a \over b})$$
```

vous aurez :

$$(\frac{a}{b})$$

Puisque les parenthèses ne sont pas dans un contexte de délimiteur, elles ne sont *pas* élargie.

Les délimiteurs vont par paires : Un délimiteur ouvrant à la gauche de la sous-formule et un délimiteur fermant à sa droite. Vous pouvez choisir explicitement une hauteur plus grande pour un délimiteur avec les commandes `\bigl`, `\bigr`, et leurs pendantes (p. 219)[4]. Par exemple, pour avoir la formule affichée :

$$\bigl(f(x) - x\bigr)\bigl(f(y) - y\bigr)$$

dans laquelle les parenthèses externes sont un peu plus grandes que leurs internes, vous devez écrire :

```
$$\bigl( f(x) - x \bigr) \bigl( f(y) - y \bigr)$$
```

Les 22 délimiteurs plain TEX, montré à leur taille normale, sont :

( ) [ ] { } ⌊ ⌋ ⌈ ⌉ ⟨ ⟩ / \ | ‖ ↑ ↓ ↕ ⇑ ⇓ ⇕

Voici les plus grandes tailles fournies explicitement par plain TEX (les versions `\Biggl`, `\Biggr`, etc.) :

$$\Biggl(\Biggr) \Biggl[\Biggr] \Biggl\{ \Biggr\} \Biggl\lfloor \Biggr\rfloor \Biggl\lceil \Biggr\rceil \Biggl\langle \Biggr\rangle \Biggl/ \Biggr\backslash \Biggl\| \Biggr\| \Biggl\uparrow \Biggr\downarrow \Biggl\updownarrow \Biggr\Uparrow \Biggl\Downarrow \Biggr\Updownarrow$$

Les délimiteurs (sauf pour '`(`', '`)`', et '`/`') sont les symboles listés sur les pages 199–200. Ils sont listés à un seul endroit sur la page 146 de *The TEXbook* et 999 de la traduction française.

Un délimiteur peut appartenir à n'importe quelle classe. Pour un délimiteur que vous agrandissez avec `\bigl`, `\bigr`, etc., la classe est déterminée par la commande : "opener" pour les commandes-`l`, "closer" pour les commandes-`r`, "relation" pour les commandes-`m`, et "ordinary symbol" pour les commandes-`g`, c'est-à-dire, `\Big`.

Vous pouvez obtenir un délimiteur de deux différentes manières :

1) Vous pouvez rendre un caractère délimiteur en lui assignant un code de délimiteur non négatif (voir plus bas) avec la commande `\delcode` (p. 260). Néanmoins le caractère n'agit comme délimiteur que si vous l'utilisez dans un contexte de délimiteur[5].
2) Vous pouvez produire un délimiteur explicitement avec la commande `\delimiter` (p. 213), en analogie avec la manière par laquelle vous pouvez produire un caractère ordinaire avec la commande `\char` ou

[4] Plain TEX défini les différentes commandes `\big` en utilisant `\left` et `\right` pour obtenir un contexte de délimiteur. Il règle la taille en construisant une formule vide de la hauteur désirée.

[5] Il est possible d'utiliser un caractère avec un code de délimiteur non négatif dans un contexte où il n'est pas un délimiteur. Dans ce cas TEX ne fait pas la recherche ; à la place, il utilise simplement le caractère de façon ordinaire (voir la page 156 de *The TEXbook* et 999 de la traduction française).

un caractère mathématique avec la commande `\mathchar`. La commande `\delimiter` utilise les mêmes code de délimiteur que ceux utilisés dans une entrée de table `\delcode`, mais avec un chiffre supplémentaire devant pour indiquer une classe. Il est rare d'utiliser `\delimiter` en dehors d'une définition de macro.

Un code de délimiteur dit à TEX comment chercher un caractère de sortie approprié pour représenter un délimiteur. Les règles de cette recherche sont assez compliquées (voir les pages 156 et 442 de *The TEXbook* et 999 et 999 de la traduction française). Une compréhension complète de ces règles requière des connaissances sur l'organisation des fichiers de métriques des polices, un sujet qui n'est pas seulement en dehors de l'objectif de ce livre mais également en dehors de l'objectif de *The TEXbook*.

En résumé la recherche travaille comme cela. Le code de délimiteur spécifie un "petit" caractère de sortie et un "grand" caractère de sortie en fournissant une position dans la police et une famille de police pour chacun (see p. 260). En utilisant cette information, TEX peut trouver (ou construire) de plus en plus grandes versions du délimiteur. TEX essaye d'abord différentes tailles (de la petite à la grande) du "petit" caractère dans la "petite" police et ensuite différentes tailles (là aussi de la petite à la grande) du "grand" caractère dans la "grande" police, en cherchant un dont la hauteur plus la profondeur soit suffisamment grande. Si aucun de ces caractères trouvé n'est assez grand, il utilise le plus grand. Il est possible que le petit caractère, le grand caractère, ou les deux ait été laissé non spécifié (indiqué par un zéro dans la partie appropriée du code de délimiteur). Si seulement un caractère a été spécifié, TEX l'utilise. Si aucun ne l'a été, il remplace le délimiteur par un espace de largeur `\nulldelimiterspace`.

démérites. TEX utilise des démérites comme une mesure de combien une ligne n'est pas désirable quand il coupe un paragraphe en lignes (voir "coupure de ligne", p. 59). Les démérites d'une ligne sont affectés par la médiocrité de la ligne et par les pénalités associées avec la ligne. Le but de TEX dans le choix d'un arrangement particulier de lignes est de minimiser le total des démérites pour le paragraphe, qu'il calcule en additionnant les démérites des lignes individuelles. voir les pages 97–98 de *The TEXbook* et 99–99 de la traduction française pour les détails sur comment TEX coupe un paragraphe en lignes. TEX n'utilise pas de démérites quand il choisit des coupures de page ; à la place, il utilise une mesure similaire appelée le "coût" d'une coupure de page particulière.

élément. Le terme *élément* est souvent utilisé pour faire référence à un composant d'une liste horizontale, verticale ou mathématique, c'est-à-dire, une liste d'éléments que TEX construit quand il est dans un mode horizontal, vertical ou mathématique.

élément extraordinaire. Un *élément extraordinaire* est un élément d'information qui dit à TEX d'effectuer une action qui n'entre pas dans le schéma ordinaire des choses. un élément extraordinaire peut apparaître dans une liste horizontale ou verticale, comme une boîte ou un élément ressort. TEX compose un élément extraordinaire comme une boîte de largeur, hauteur et profondeur à zéro—en d'autre termes, une boîte qui ne contient rien et n'occupe pas d'espace.

Trois sortes d'éléments extraordinaires sont construit dans TEX :

- Les commandes `\openout`, `\closeout` et `\write` (p. 257) produit un élément extraordinaire pour agir sur un fichier de sortie. TEX diffère l'opération jusqu'a ce qu'il envoi la prochaine page dans le fichier `.dvi` (à moins que l'opération soit précédée par `\immediate`). TEX utilise un élément extraordinaire pour ces commandes parce que elles n'ont rien à faire avec ce qu'il compose quand il les rencontre.
- La commande `\special` (p. 258) demande à TEX d'insérer un certain texte directement dans le fichier `.dvi`. comme avec la commande `\write`, TEX diffère l'opération jusqu'a ce qu'il envoi la prochaine page dans le fichier `.dvi` Une utilisation typique de `\special` serait de nommer un fichier graphique que le driver d'affichage pourrait incorporer dans votre sortie finale.
- Quand vous changez de langages avec les commandes `\language` ou `\setlanguage` (p. 134), TEX insère un élément extraordinaire qui l'instruit de l'usage d'un certain jeu de règle de césure à suivre quand il coupe un paragraphe en lignes.

Une implémentation particulière de TEX peut procurer des éléments extraordinaires additionnels.

entête. Un *entête* est un matériel que TEX met en haut de chaque page, Au-dessus du texte de cette page. L'entête pour un simple rapport peut être constitué du titre du coté gauche de la page et du texte "Page *n*" du coté droite de la page. D'habitude, un entête est constitué d'une seule ligne, que vous pouvez mettre en assignant une liste de tokens à `\headline` (p. 149). L'entête par défaut de plain TEX est à blanc. Il est aussi possible de produire des entête multiligne ; voir page 282 pour voir comment faire.

espace. Vous pouvez demander à TEX de mettre un *espace* entre deux éléments de plusieurs façon :

1) Vous pouvez écrire quelque chose que TEX traite comme un token espace : un ou plusieurs caractères blancs, La fin d'une ligne (le caractère fin de ligne agit comme un espace) ou toute commande qui se développe en token espace. TEX généralement traite plusieurs espaces consécutifs comme équivalent à un seul, en incluant le cas où les espaces comprennent une seule fin de ligne. (Une ligne vide indique la fin d'un paragraphe ; il fait que TEX génère un token

`\par`.) TEX ajuste la taille de cette sorte d'espace en fonction de la longueur requise par le contexte.

2) Vous pouvez écrire une commande de saut qui produit le ressort que vous spécifie dans la commande. Le ressort peut s'étirer ou se rétrécir, produisant plus ou moins d'espace. Vous pouvez avoir un ressort vertical comme un ressort horizontal. Le ressort disparaît à chaque fois qu'il est après une coupure de ligne ou de page.
3) Vous pouvez écrire un crénage. Un crénage produit un montant d'espace fixe qui ne s'étire ni se rétrécit et ne disparaît pas sur un une coupure de ligne ou de page (à moins d'être suivi immédiatement par un ressort). L'usage le plus courant du crénage est d'établir une relation spatiale fixe entre deux boîtes adjacentes.

Le ressort et les crénages peuvent avoir des valeurs négatives. Un ressort négatif ou un crénage négatif entre des éléments adjacents rapproche ces éléments l'un de l'autre.

étirement. Voir "ressort" (p. 93).

famille. Une *famille* est un groupe de trois polices reliées utilisées quand TEX est en mode mathématique. En dehors du mode mathématique, les familles n'ont aucun effet. Les trois polices d'une famille sont utilisées pour les symboles normaux (taille de texte), indices et exposants (taille script) et sous-indices, sur-exposants, etc. (taille scriptscript). Par exemple, le nombre '2' composé dans ces trois polices vous donnera '2', '$_2$' et '$_2$' (en plain TEX). Normalement vous fixerez les trois polices d'une famille à différents taille du même type de police, mais rien ne vous empêche d'utiliser des types différents pour les trois polices ni d'utiliser la même police deux fois dans une famille.

TeX procure jusqu'a seize familles, numérotées de 0 à 15. Par exemple, la famille 0 en plain TEX consiste en roman 10 points pour le texte, roman 7 points pour le script et roman 5 points pour le scriptscript. Plain TEX défini aussi la famille 1 qui est constituée des polices mathématiques italiques et réserve les familles 2 et 3 pour respectivement les symboles spéciaux et les extensions mathématiques[6]. Si vous devez définir une famille vous même, vous devrez utiliser la commande `\newfam` (p. 252) pour avoir le numéro d'une famille qui n'est pas utilisée et les commandes `\textfont`, `\scriptfont` and `\scriptscriptfont` commands (p. 219) pour assigner des polices à cette famille.

fichier. Un *fichier* est un flot d'information que TEX interprète ou crée. Des fichiers sont gérés par le système d'exploitation qui supervise votre exécution de TEX. TEX manipule des fichiers dans quatre contextes différent :

[6] Les familles 2 et 3 sont spéciales en ce que leurs fichiers de métriques de police doivent inclure des paramètres pour l'espacement mathématique.

1) Un "fichier source" est celui que TEX lit avec ses "yeux" (voir "Anatomie de TEX", p. 48) et interprète en accord avec ses règles ordinaires. Votre fichier d'entrée primaire—celui que vous spécifiez après '**' ou sur la ligne de commande quand vous invoquez TEX—est un fichier source, et ainsi pour tout fichier que vous appelez avec une commande `\input` (p. 255).
2) Un "fichier résultat" est celui qui contient les résultats de l'exécution de TEX. TEX crée deux fichiers résultat : Le fichier `.dvi` et le fichier log. Le fichier `.dvi` contient l'information nécessaire pour imprimer votre document ; Le fichier log contient un enregistrement de ce qui s'est passé pendant l'exécution, incluant tout messages d'erreur que TEX a généré. Si votre fichier source primaire s'appelle `screed.tex`, vos fichiers `.dvi` et log se nommeront `screed.dvi` et `screed.log`[7].
3) Pour lire un fichier avec la commande `\read` (p. 256) vous devez associer le fichier avec un flot d'entrée. Vous pouvez avoir jusqu'a 16 flots d'entrée actif à la fois, numéroté de 0 à 15. La commande `\read` lit un seul fichier et le fait avec la valeur d'une séquence de contrôle désignée, donc lire avec `\read` est très différent de lire avec `\input` (qui apporte un fichier entier). TEX prend tout flot d'entre qui n'est pas numéroté entre 0 et 15 comme référence au terminal, donc '`\read16`', disons, lit le prochaine ligne que vous saisissez sur le terminal.
4) Pour écrire dans un fichier avec la commande `\write` (p. 258) vous devez associer le fichier avec un flot de sortie. Vous pouvez avoir jusqu'a 16 flots de sortie actifs à la fois, numéroté de 0 à 15. les flots d'entrée et de sortie sont indépendant. Tout ce qui est envoyé vers un flot de sortie avec un numéro négatif va vers le fichier log ; tout ce qui est envoyé vers un flot de sortie avec un numéro supérieur à 15 va à la fois vers le fichier log et le terminal. Ainsi '`\write16`', disons, écrit une ligne sur le terminal et aussi envoie cette ligne vers le fichier log.

Vous devez ouvrir un fichier de flot avant de pouvoir l'utiliser. Un fichier de flot d'entrée est ouvert avec une commande `\openin` (p. 256) et un fichier de flot de sortie est ouvert avec une commande `\openout` (p. 257). Par propreté, vous devez fermer un fichier de flot quand vous en avez fini avec lui, néanmoins TEX le fera à la fin du traitement si vous ne l'avez pas fait. Les deux commandes pour fermer un fichier de flot sont `\closein` (p. 256) et `\closeout` (p. 257). Un avantage de fermer un flot quand vous en avez fini avec lui est que vous pouvez le réutiliser le flot pour un autre fichier. Ceci peut être essentiel quand vous lisez un longue suite de fichiers.

De plus vous pouvez assigner des numéros vous-même aux flot d'entrée et de sortie, Il est préférable de le faire avec les commandes `\newread` et `\newwrite` (p. 252). Vous pouvez avoir plus d'un flot associé avec un

[7] C'est la convention usuelle, mais des implémentations particulières de TEX sont libres de changer cela.

seul fichier, mais vous aurez des saletés (probablement non diagnostiqué) à moins que tous les flots soit des flots d'entrée. Associer plus d'un flot avec un fichier d'entrée peut-être utile quand vous voulez utiliser le même fichier d'entrée pour deux usages différents.

D'habitude, TEX diffère les actions d'ouverture, d'écriture et de fermeture d'un flot de sortie jusqu'a ce qu'il enregistre une page avec `\shipout` (voir la page 227 de *The TEXbook* et 999 de la traduction française pour les détails). Cette propriété s'applique même aux messages écrit sur le terminal avec `\write`. Mais vous pouvez demande à TEX pour faire une action sur un flot de sortie immédiatement en faisant précéder la commande d'action de `\immediate` (p. 258). Par exemple :

```
\immediate\write16{Do not pass GO! Do not collect $200!}
```

fichier format. Un *fichier format* est un fichier qui contient une image de la mémoire de TEX sous la forme dans laquelle il peut être rechargé rapidement. Un fichier format peut être créé avec la commande `\dump` (p. 271). L'image contient un enregistrement complet des définitions (de polices, macros, etc.) qui étaient présentes quand le dump a été effectué. En utilisant `virtex`, un forme spéciale "vierge" de TEX, vous pouvez recharger le fichier format à haute vitesse et continuer dans le même état dans lequel était TEX au moment du dump. L'avantage du fichier format sur un fichier d' entrée ordinaire contenant la même information est que TEX peut la charger beaucoup plus vite.

Des fichiers de format doivent être créés que `initex`, une forme spéciale de TEX écrite à cette intention. Ni `virtex`, ni `initex` n'ont d'outils autres que les primitives construite dans le programme TEX lui-même.

Une forme préchargée de TEX est celle qui a un fichier format conforme chargé et est prêt à accepter l'entrée de l'utilisateur. La forme de TEX qui se nomme `tex` a souvent les définitions plain TEX préchargées. (Plain TEX est normalement fourni sous deux autres formes : comme une fichier de format et comme une fichier source de TEX. Dans certain environnements, `tex` est équivalent à appeler `virtex` et charge alors `plain`.) Créer des formes préchargées de TEX nécessite un programme spécial ; il ne peut pas être fait en n'utilisant que les outils de TEX lui-même.

fichier log. Voir "fichier" (p. 67).

filets. Un *filet* est un rectangle plein noir. Un filet, comme une boîte, a une largeur, une hauteur et une profondeur. La dimension vertical du rectangle est la somme de sa hauteur et de sa profondeur. Une ligne droite horizontale ou verticale est un cas spécial de filet.

Un filet peut être soit horizontal, soit vertical. La distinction entre un filet horizontal et un vertical diffère selon la façon de produire le filet, un filet vertical peut être court et épais (et donc ressembler à une ligne horizontale), tandis qu'un filet horizontal peut être grand et maigre (et donc ressembler à une ligne verticale). la notion de filet de TEX est plus

générale que celle des typographes, qui pensent à un filet comme à une ligne et ne veulent habituellement pas appeler filet une boîte noire carrée.

Vous pouvez produire un filet horizontal en utilisant la commande `\hrule` et un filet vertical en utilisant la commande `\vrule` (p. 178). Par exemple, la séquence de contrôle `\hrule` par elle-même produit un filet fin qui traverse la page, comme ceci :

La commande '`\vrule height .25in`' produit un filet vertical qui s'allonge de .25 pouces sur la page comme ceci :

Il y a deux différences entre les filets horizontaux et verticaux :

1) Pour un filet horizontal, TEX prend comme largeur par défaut la largeur de la plus petite boîte ou alignement qui l'englobe. Pour un filet vertical, TEX prend comme hauteur et profondeur par défaut de la même façon. (Le défaut est la taille que vous obtenez si vous ne donnez pas de taille explicitement pour cette dimension.)
2) Un filet horizontal est un élément fondamentalement vertical qui ne peut participer à une liste horizontale, tandis qu'un filet vertical est un élément horizontal qui ne peut participer à une liste verticale. Cette propriété peut sembler étrange d'un premier abord, mais il y a une bonne raison à cela : Un filet horizontal s'étend visuellement de gauche à droite et donc sépare des éléments d'une liste verticale, tandis qu'un filet vertical s'étend visuellement de haut en bas et ainsi sépare des éléments d'une liste horizontale.

Si vous construisez un filet avec trois dimensions explicites, cela donnera la même chose que vous fassiez un filet horizontal ou vertical. Par exemple, la commande '`\vrule height1pt depth2pt width3in`' produit ce filet au look horizontal :

Vous trouverez un état précis du traitement des filets de TEX dans les pages 221–222 de *The TEXbook* et 999–999 de la traduction française.

flots d'entrée. Voir "fichier" (p. 67).

flots de sortie. Voir "fichier" (p. 67).

global. Une définition *globale* est effective jusqu'a la fin du document ou jusqu'a ce qu'elle soit écrasée par une autre définition, même quand elle apparaît dans un groupe. Ainsi une définition globale n'est pas affectée par les frontières de groupe. Vous pouvez rendre n'importe quelle définition globale en la préfixant avec la commande `\global` (p. 236) a moins que `\globaldefs` (p. 236) soit négatif.

Il y a un moyen spécial de rendre une définition de macro globale. Normalement vous définissez une macro en utilisant soit la commande `\def` soit la commande `\edef` (p. 238). Si vous utilisez `\gdef` ou `\xdef`

au lieu de `\def` et `\edef` respectivement, la définition de macro sera globale. Parce que, '`\gdef`' est équivalent à '`\global\def`' et '`\xdef`' est équivalent à '`\global\edef`'.

groupe. Un *groupe* est une partie de votre manuscrit que TEX traite comme une unité. Vous indiquez un groupe en l'entourant avec les accolades '`{`' et '`}`' (ou tout autre caractères avec le code de catégorie approprié).

La plus importante propriété d'un groupe est que tout définition ou assignation non globale que vous faites dans un groupe disparaît quand le groupe se termine. Par exemple, si vous écrivez :

```
Please don't pour {\it any} more tea into my hat.
```

La séquence de contrôle `\it` demande à TEX de mettre le mot '`any`' en police italique mais n'affecte pas le reste du texte. Comme autre exemple, si vous utilisez le paramètre `\hsize` (p. 120) pour changer la longueur de la ligne dans un groupe, La longueur de ligne retourne à sa valeur précédente un fois que TEX est sorti du groupe.

Les groupes sont aussi pratiques comme moyen de contrôler l'espacement. Par exemple, si vous saisissez :

```
\TeX for the Impatient and the Outpatient too.
```

vous aurez :

TEXfor the Impatient and the Outpatient too.

puisque la séquence de contrôle `\TeX` (qui produit le logo TEX) absorbe l'espace qui le suit. vous voulez probablement ceci :

TEX for the Impatient and the Outpatient too.

Une manière d'obtenir ceci est d'englober '`\TeX`' dans un groupe :

```
{\TeX} for the Impatient and the Outpatient too.
```

L'accolade droite empêche la séquence de contrôle d'absorber l'espace.

hauteur. La *hauteur* d'une boîte est la distance à laquelle la boîte s'étend au dessus de sa ligne de base.

hbox. Une *hbox* (boîte horizontale) est une boîte que TEX construit en plaquant les articles d'une liste horizontale l'une après l'autre, de la gauche vers la droite. Une hbox, prise comme unité, n'est ni fondamentalement horizontale ni fondamentalement verticale, c'est-à-dire, qu'elle peut apparaître comme un article soit d'une liste horizontale soit d'une liste verticale. Vous pouvez construire une hbox avec la commande `\hbox` (p. 166).

insertion. Une *insertion* est une liste verticale contenant du matériel devant être inséré sur une page quand TEX a fini de construire cette page[8]. Des exemple de telles insertions sont les notes de pied de page et les figures. Les commandes plain TEX pour créer des insertions sont `\footnote`, `\topinsert`, `\midinsert`, et `\pageinsert`, aussi bien que la commande primitive `\insert` elle-même (pp. 151–154). Le mécanisme de TEX pour manipuler des insertions est plus compliqué ; voir les pages 122–125 de *The TEXbook* et 999–999 de la traduction française pour les détails.

largeur. La *largeur* d'une boîte est le montant d'espace horizontal qu'il occupe, c'est-à-dire, la distance de son coté gauche à son coté droit. Le matériel composé dans une boîte peut être plus large que la boîte elle-même.

ligature. Une *ligature* est un simple caractère qui remplace une séquence particulière de caractères adjacents dans un document composé. Par exemple, le mot '`office`' est composé "office" et non "office", par les systèmes de composition de haute qualité. La connaissance des ligatures est construite dans les polices que vous utilisez, donc vous n'avez rien à faire pour que TEX les produisent. (Vous pouvez empêcher la ligature d'"office", comme nous l'avons dit plutôt, en saisissant '`of{f}ice`' dans votre entrée.) TEX est aussi capable d'utiliser son mécanisme de ligature pour composer la première ou la dernière lettre d'un mot différemment de la même lettre qui apparaît au milieu d'un mot Vous pouvez empêcher cet effet (si vous le rencontrez) en utilisant la commande `\noboundary` (p. 107).

Parfois, vous aurez besoin d'une ligature d'une langue européenne. TEX ne les produit pas automatiquement sauf si vous utilisez une police dessinée pour cette langue. Un certain nombre de ces ligatures, par exemple, Æ, sont accessible par une commande (voir "Lettres et ligatures pour alphabets européens", p. 103).

ligne de base. La *ligne de base* d'une boîte est une ligne imaginaire qui traverse la boîte. Quand TEX assemble les boîtes d'une liste horizontale en une boîte plus grande, il aligne les boîtes de la liste pour que leurs lignes de base coïncident. Par analogie, pensez que vous écrivez sur un bloc de papier ligné. Chaque lettre que vous écrivez a une lignes de base implicite. Pour aligner les lettres horizontalement, vous les placez sur le

[8] TEX lui-même n'insère pas le matériel—il rend juste le matériel accessible à la routine de sortie, qui est alors responsable de la transférer sur la page composée. Le seul effet immédiat de la commande `\insert` (p. 153) est de changer les calculs de coupure de page de TEX pour qu'il laisse de la place sur la page pour le matériel à insérer. Plus tard, quand TEX coupera réellement la page, il divisera le matériel à insérer en deux groupes : le matériel qui passe sur la page courante et le matériel qui ne passe pas. Le matériel qui passe sur la page est placé dans des registres de boîte, un par insertion et le matériel qui ne passe pas est transférer sur la page suivante. Cette procédure autorise TEX de faire des choses comme distribuer des parties d'une longue note de pied de page sur plusieurs pages consécutives.

bloc pour que leur lignes de base s'ajuste avec les lignes-guides claires imprimées sur le bloc.

Une boîte peut et souvent doit s'étendre sous sa ligne de base. Par exemple, la lettre 'g' s'étend sous la ligne de base de sa boîte parce qu'il a une descendante (la boucle du dessous du 'g').

liste. Une *liste* est une séquence d'éléments (boîtes, ressorts, crénages, etc.) cela comprend tout le contenu d'une hbox, d'une vbox ou d'une formule mathématique. Voir "liste horizontale" (p. 73), "liste verticale" (p. 73).

liste horizontale. Une *liste horizontale* est une liste d'articles que TEX a produit quand il était dans un de ses modes horizontaux, c'est-à-dire, en assemblant soit un paragraphe soit une hbox. Voir "mode horizontal" ci-dessous.

liste verticale. Une *liste verticale* est une liste d'éléments que TEX a produit quand il était dans une de ses modes vertical, c'est-à-dire, en assemblant soit une vbox soit une page. Vois "mode vertical" ci-dessous.

macro. Une *macro* est une définition qui donne un nom à un modèle de texte d'entrée de TEX[9]. Le nom peut être soit une séquence de contrôle soit un caractère actif. Le modèle est appelé le "texte de remplacement". La commande primaire pour définir des macros est la séquence de contrôle `\def`.

Comme exemple simple, supposez que vous ayez un document dans lequel la séquence '$\cos\theta+i\sin\theta$' apparaît plusieurs fois. Au lieu de l'écrire à chaque fois, vous pouvez définir une macro pour cela :

```
\def\arctheta{\cos \theta + i \sin \theta}
```

Maintenant à chaque fois que vous avez besoin de cette séquence, vous pouvez simplement "appeler" la macro en écrivant '`\arctheta`' et vous l'aurez. Par exemple, '`$e^{\arctheta}$`' vous donnera '$e^{\cos\theta+i\sin\theta}$'.

Mais la réelle puissance des macros tient au fait qu'une macro peut avoir des paramètres. Quand vous appelez une macro qui a des paramètres, vous fournissez des arguments qui se substituent à ces paramètres. Par exemple, supposez que vous écrivez :

```
\def\arc#1{\cos #1 + i \sin #1}
```

La notation `#1` désigne le premier paramètre de la macro, qui dans ce cas n'a qu'un paramètre. Vous pouvez maintenant produire une forme similaire, telle que '$\cos 2t+i\sin 2t$', avec l'appel de macro '`\arc{2t}`'.

Plus généralement, une macro peut avoir jusqu'a neuf paramètres, que vous désignez par '`#1`', '`#2`', etc. dans la définition de la macro. TEX

[9] Plus précisément, La définition donne un nom à une séquence de tokens.

fourni deux types de paramètres : les paramètres délimités et les paramètres non délimités. Brièvement, un paramètre délimité a un argument qui est délimité ou terminé par une séquence spécifique de tokens (le délimiteur), tandis qu'un paramètre non délimité a un argument qui n'a pas besoin de délimiteur pour le terminer. Premièrement, nous expliquerons comment fonctionnent les macros quand elles n'ont que des paramètres non délimités, et ensuite, nous expliquerons comment elles fonctionnent quand elles ont des paramètres délimités.

Si une macro n'a que des paramètres non délimités, ces paramètres doivent apparaître l'un après l'autre dans la définition de macro *sans rien entre eux ou entre le dernier paramètre et l'accolade gauche au début du texte de remplacement*. Un appel d'une telle macro consiste en le nom de la macro suivi par les arguments de l'appel, un pour chaque paramètre. Chaque argument est soit :

- un token simple autre qu'un accolade gauche ou droite ou
- une séquence de tokens englobé entre une accolade gauche et une accolade droite correspondante[10].

Quand TEX rencontre une macro, il développe la macro dans son œsophage (voir "Anatomie de TEX", p. 48) en substituant chaque argument par le paramètre correspondant dans le texte de remplacement. Le texte résultant peut contenir d'autre appels de macro. Quand TEX rencontre un tel appel de macro imbriqué, il développe cet appel immédiatement sans regarder ce qui suit l'appel [11]. Quand l'œsophagede TEX tombe sur une commande primitive qui ne peut être développée plus loin, TEX passe cette commande à l'estomac de TEX. L'ordre de développement est parfois critique, donc de façon à vous aider à le comprendre, nous vous donnerons un exemple de TEX au travail.

Supposez que vous procurez à TEX l'entrée suivante :

```
\def\a#1#2{\b#2#1\kern 2pt #1}
\def\b{bb}
\def\c{\char49 cc}
\def\d{dd}
\a\c{e\d} % Call on \a.
```

Alors l'argument correspondant à `#1` est `\c`, et l'argument correspondant à `#2` est `e\d`. TEX développe l'appel de macro d'après les étapes suivantes :

```
\b e\d\c\kern 2pt \c
bbe\d\c\kern 2pt \c
\d\c\kern 2pt \c   ('b', 'b', 'e' envoyés vers l'estomac)
dd\c\kern 2pt \c
\c\kern 2pt \c   ('d', 'd' envoyés vers l'estomac)
```

[10] L'argument peut avoir des paires d'accolades englobantes et chacune de ces paires peut désigner soit un groupe soit un autre argument de macro.

[11] En terminologie informatique, le développement est "profondeur d'abord" plutôt que "largeur d'abord". Notez que vous pouvez modifier l'ordre du développement avec des commandes telles que `\expandafter`.

```
\char49 cc\kern 2pt \c
\c   (‘\char’, ‘4’, ‘9’, ‘c’, ‘c’, ‘\kern’, ‘2’, ‘p’, ‘t’ envoyés vers l'estomac)
\char49 cc
(‘\char49’, ‘c’, ‘c’ envoyés vers l'estomac)
```

Notez que les lettres ‘`b`’, ‘`c`’, ‘`d`’ et ‘`e`’ et les séquences de contrôle ‘`\kern`’ et ‘`\char`’ sont toutes des commandes primitives qui ne peuvent être développées.

Une macro peut aussi avoir des “paramètres délimités”, qui peuvent être mélangés avec des non délimités dans toutes les combinaisons. L'idée d'un paramètre délimités est que TEX trouve l'argument correspondant en recherchant une certaine séquence de tokens qui marquent la fin de l'argument—le délimiteur. Cela fait, quand TEX recherche un tel argument, il prend comme argument tout les tokens à partir de la position courante de TEX mais sans inclure le délimiteur.

Vous indiquez un paramètre délimité en écrivant ‘`#`n’ (n doit être entre 0 et 9) suivi par un ou plusieurs tokens qui agissent comme le délimiteur. Le délimiteur s'étend jusqu'au prochains ‘`#`’ ou ‘`{`’—qui ont un sens car ‘`#`’ débute un autre paramètre et ‘`{`’ le texte de remplacement.

Le délimiteur ne peut être ‘`#`’ ou ‘`{`’, donc vous pouvez appeler un paramètre délimité à partir d'un non délimité en recherchant ce qui le suit.

Si le caractère après le paramètre est ‘`#`’ ou ‘`{`’, vous avez un paramètre non délimité ; autrement il sera délimité. Notez la différence dans les arguments pour les deux sortes de paramètres—un paramètre non délimité est désigné soit par un simple token, soit par une séquence de tokens englobés entre des accolades, tandis qu'un paramètre délimité est désigné par n'importe que nombre de tokens, même aucun.

Un exemple de macro utilisant deux paramètres délimités est :

```
\def\diet#1 #2.{On #1 we eat #2!}
```

Ici le premier paramètre est délimité par un simple espace et le second paramètre est délimité par un point. Si vous saisissez :

```
\diet Tuesday turnips.
```

vous obtiendrez le texte “On Tuesday we eat turnips!”. Mais si les tokens délimitants sont englobés dans un groupe, TEX ne les considèrent pas délimités. Donc si vous écrivez :

```
\diet {Sunday mornings} pancakes.
```

vous obtiendrez le texte ‘On Sunday mornings we eat pancakes!’ même s'il y a un espace entre ‘`Sunday`’ et ‘`morning`’. Quand vous utilisez un espace comme délimiteur, une caractère fin de ligne délimite aussi normalement l'argument car TEX convertit la fin de ligne en espace avant que le mécanisme de macro ne le voit.

Il arrivera que vous ayez à définir une macro qui ait ‘`#`’ comme caractère significatif. Vous aurez plus de chance d'avoir besoin de faire cela quand vous définirez une macro qui à son tour définit une seconde macro. Comment alors faire à propos des paramètres de la seconde macro sans mettre

TEX dans la confusion ? La réponse est que vous saisissez deux '`#`' pour chacun de ceux que vous voulez quand la première macro est développée. Par exemple, supposons que vous écriviez la définition de macro :

```
\def\first#1{\def\second##1{#1/##1}}
```

Alors l'appel '`\first{One}`' défini '`\second`' comme :

```
\def\second#1{One/#1}
```

et l'appel consécutif '`\second{Two}`' produit le texte 'One/Two'.

Un certain nombre de commandes fournissent des moyens supplémentaires pour définir des macros (voir pp. 238–250). Pour les règles complètes concernant les macros, voir le chapitre 20 de *The TEXbook* et de la traduction française.

magnification. Quand TEX compose un document, il multiplie toutes les dimensions par un facteur de magnification $f/1000$, où f est la valeur du paramètre `\mag` (p. 231). Puisque la valeur par défaut de `\mag` est 1000, le cas normal est que votre document est composé comme il est spécifié. Augmenter la magnification est souvent utile quand vous composez un document qui sera réduit photographiquement.

Vous pouvez aussi appliquer de la magnification sur une seule police pour avoir une version plus petite ou plus grande de cette police que sa "taille d'origine". Vous devez procurer au driver d'impression unfichier de forme (voir "police", p. 88) pour chaque magnification de police que vous utilisez—a moins que la police soit construite dans votre imprimante et que votre driver d'impression la connaisse. Quand vous définissez une police avec la commande `\font` (p. 229), vous pouvez spécifier une magnification avec le mot '`scaled`'. Par exemple :

```
\font\largerbold = cmbx10 scaled 2000
```

définit '`\largerbold`' comme une police qui est deux fois plus grande que `cmbx10` (Computer Modern Grasse Etendue en 10 points) et a les formes de caractère uniformément agrandies d'un facteur de 2.

De nombreux centres informatiques trouvent pratique de fournir des polices agrandies par un ratio de 1.2, correspondant à des valeurs de magnification de 1200, 1440, etc. TEX a des noms spéciaux pour ces valeurs : '`\magstep1`' pour 1200, '`\magstep2`' pour 1440, et ainsi de suite jusqu'a '`\magstep5`'. La valeur spéciale '`\magstephalf`' correspond à une magnification de $\sqrt{1.2}$, qui est visuellement à mi-chemin entre '`\magstep0`' (pas de magnification) et '`\magstep1`'. Par exemple :

```
\font\bigbold = cmbx10 scaled \magstephalf
```

Vous pouvez spécifier une dimension comme elle sera mesurée dans le document final indépendamment des magnifications en mettant '`true`' devant l'unité. Par exemple, '`\kern 8 true pt`' produit un crénage de 8 points quelque soit la magnification.

marges. Les marges d'une page définissent un rectangle qui normalement contient la matière imprimée sur la page. Vous pouvez demander à TEX d'imprimer du matériel en dehors de ce rectangle, mais uniquement en demandant des actions explicites qui déplacent le matériel à cet endroit. TEX considère les entêtes et les pieds de page comme étant en dehors des marges.

Le rectangle est défini en fonction de son coin en haut à gauche, de sa largeur, et de sa profondeur. l'emplacement du coin en haut à gauche est défini par les paramètres `\hoffset` et `\voffset` (p. 146). L'usage est de placer ce coin à un pouce du haut et à un pouce du bord gauche de la page, correspondant à une valeur de zéro pour `\hoffset` et `\voffset`[12]. La largeur du rectangle est donné par `\hsize` et la profondeur par `\vsize`.

Les implications de ces conventions sont :

- La marge gauche est donnée par `\hoffset`+`1in`.
- La marge droite est donnée par la largeur du papier moins `\hoffset` + `1in` + `\hsize`.
- La marge du haut est donnée par `\voffset`+`1in`.
- La marge du bas est donnée par la longueur du papier moins `\voffset` + `1in` + `\vsize`.

A partir de ces informations vous pouvez voir quels paramètres vous devez changer pour modifier les marges.

Chaque changements que vous faites à `\hoffset`, `\voffset` ou `\vsize` prendra effet la prochaine fois que TEX débutera une page. En d'autres termes, si vous les changez dans une page, le changement n'affectera que la page *suivante*. Si vous changez `\hsize`, le changement deviendra effectif immédiatement.

marque. Une *marque* est un élément que vous pouvez insérer dans une liste horizontale, verticale ou mathématique et plus tard la retrouver dans votre routine de sortie. Les marques sont utiles pour des usages tels que garder une trace des sujets apparents dans des entêtes de page. Chaque marque a une liste de tokens—le "texte de marque"—associé avec elle. La commande `\mark` (p. 150) attend une telle liste de token comme son argument, et attend un élément contenant cette liste de token (après expansion) pour toute liste que TEX est en train de construire. Les commandes `\topmark`, `\firstmark` et `\botmark` (p. 150) peuvent être utilisées pour retrouver diverses marques sur une page. Ces commandes sont le plus souvent utilisées dans les entêtes et les pieds de page.

Voici un exemple simplifié. Supposez que vous définissiez une macro d'entête de section comme suit :

```
\def\section#1{\medskip{\bf#1}\smallskip\mark{#1}}
% #1 is the name of the section
```

[12] Il nous semble qu'il s'agisse d'une convention bizarre. Il aurait été plus naturel d'avoir le point $(0, 0)$ pour `\hoffset` et `\voffset` dans le coin en haut à gauche du papier et d'avoir leur valeur par défaut à un pouce.

Cette macro, quand elle est appelée, produira un entête de section en caractère gras et enregistrera aussi le nom de la section comme marque. Vous pouvez maintenant définir l'entête de chaque page imprimée comme suit :

```
\headline = {\ifodd\pageno \hfil\botmark\quad\folio
   \else \folio\quad\firstmark\hfil \fi}
```

Chaque page paire (celle de gauche) aura maintenant le numéro de page suivi par le nom de la première section de cette page, tandis que chaque page impaire (celle de droite) aura le numéro de page suivi du nom de la dernière section de cette page. Les cas spéciaux, par exemple, pas de sections commençant sur une page, s'affichera généralement correctement grâce à la manière dont travaillent `\firstmark` et `\botmark`.

Quand vous partagez une page en utilisant la commande `\vsplit` (p. 155) vous pouvez retrouver les textes de la première et de la dernière marque de la portion non partagée avec les commandes `\splitfirstmark` et `\splitbotmark` (p. 150).

Voir les pages 258–260 de *The TEXbook* et 999–999 de la traduction française pour une explication plus précise sur la façon de créer et retrouver des marques.

mathcode. Un *mathcode* est un nombre que TEX utilise pour identifier et décrire un caractère mathématique, c'est-à-dire, un caractère qui a un rôle particulier dans une formule mathématique. Un mathcode transmet trois sortes d'information sur un caractère : sa position dans la police, sa famille et sa classe. chacun des 256 caractères entrés possibles a un mathcode, qui est défini par le programme TEX mais peut être changé.

TEX a seize familles de police, numérotées 0–15. Chaque famille contient trois polices : une pour la taille texte, une pour la taille script et une pour la taille scriptscript. TEX choisi la taille d'un caractère particulier et donc sa police, en fonction du contexte. La classe d'un caractère spécifie son rôle dans une formule (voir la page 154 de *The TEXbook* et 999 de la traduction française). Par exemple, le signe égal '`=`' est dans la classe 3 (Relation). TEX utilise sa connaissance des classes de caractère quand il décide combien d'espace mettre entre différent composants d'une formule mathématique.

La meilleur façon de comprendre tout ce que sont les mathcodes est de voit comment TEX les utilise. Donc nous vous montrerons ce que TEX fait avec un token de caractère t de code de catégorie 11 ou 12 dans une formule mathématique :

1) Il recherche le mathcode du caractère.
2) Il détermine une famille f à partir du mathcode.
3) Il détermine la taille s à partir du contexte.
4) Il sélectionne une police F en prenant la police de taille s dans la famille f.
5) Il détermine un numéro de caractère n à partir du mathcode.

6) Il sélectionne comme caractère c devant être composé le caractère à la position n de la police F.
7) Il ajuste l'espacement autour de c en fonction de la classe de t et du contexte ambiant.
8) Il compose le caractère c.

La dépendance du contexte dans les étapes (3) et (7) implique que TEX ne peut pas composer un caractère mathématique tant qu'il n'a pas vu tout la formule contenant le caractère mathématique. Par exemple, dans la formule '`$a\over b$`', TEX ne sait pas quelle taille aura le '`a`' tant qu'il n'aura pas vu le `\over`.

Le mathcode d'un caractère est encodé en fonction de la formule $4096c+256f+n$, où c est la classe du caractère, f sa famille et n son code de caractère ASCII dans la famille. vous pouvez changer l'interprétation de TEX pour un caractère entré dans un mode mathématique en assignant une valeur à la table d'entrée des `\mathcode` (p. 259) pour ce caractère. Le caractère doit avoir un code de catégorie à 11 (lettre) ou 12 (autre) pour que TEX recherche son `\mathcode`.

Vous pouvez définir un caractère mathématique pour avoir une famille "variable" en lui donnant une classe de 7. à chaque fois que TEX rencontre ce caractère dans une formule mathématique, il prend la famille du caractère comme étant la valeur courante du paramètre `\fam` (p. 218). Une famille variable vous autorise à spécifier la police de texte ordinaire dans une formule mathématique. Par exemple, si le caractère romain est dans la famille 0, l'assignement `\fam = 0` fera que le texte ordinaire d'une formule mathématique sera mis en type romain au lieu d'autre chose comme du type mathématique italique. Si la valeur de `\fam` n'est pas dans la fourchette de 0 à 15, TEX prend la valeur à 0, ainsi le rendu des classes 0 et 7 est équivalent. TEX fixe `\fam` à -1 a chaque fois qu'il entre en mode mathématique.

mathématique affichée. Le terme *mathématique affichée* fait référence aux formules mathématiques que TEX place sur une ligne seule avec de l'espace supplémentaire dessus et dessous pour le séparer du texte l'environnant. Une formule mathématique affichée est entourée de '`$$`'. TEX lit es mathématiques affichées dans le mode mathématiques affichées.

médiocrité. La *médiocrité* d'une ligne mesure combien les tailles des espaces inter-mot de la ligne dévient de leur valeurs naturelles, c'est-à-dire, des valeurs spécifiées dans les polices utilisées dans la ligne. Plus grande est la déviation, plus grande est la médiocrité. Similairement, la médiocrité d'une page est une mesure de combien les espaces entre les boîtes qui dessinent la page dévient de leurs valeurs idéales. (Ordinairement, la plupart de ces boîtes sont des lignes simples de paragraphes.)

Plus précisément, la médiocrité mesure de combien le ressort associé avec ces espaces doit s'étirer ou se rétrécir pour remplir la ligne ou la page exactement. TEX calcule la médiocrité comme approximativement

100 fois le cube du ratio par lequel il doit étirer ou rétrécir le ressort pour composer une ligne ou une page de la taille désirée. Par exemple, étirer le ressort par deux fois son état d'étirement signifie un ratio de 2 et une médiocrité de 800 ; la rétrécir de moitié son état de rétrécissement signifie un ratio de .5 et une médiocrité de 13. TEX traite une médiocrité plus grand que 10000 comme étant égal à 10000.

TEX utilise la médiocrité d'une ligne quand il coupe un paragraphe en lignes (voir "coupure de ligne", p. 59). Il utilise cette information en deux étapes :

1) Quand TEX choisi des coupures de ligne, il acceptera éventuellement des lignes dont la médiocrité est inférieur ou égal à la valeur de `\tolerance` (p. 128). Si TEX ne peut pas diminuer la conception d'une ligne dont la médiocrité dépasse cette valeur, il lui mettra un "underfull ou overfull hbox". TEX ne mettra un "overfull ou underfull hbox" qu'en dernier ressort, c'est-à-dire, seulement s'il n'y a pas d'autre moyen découper le paragraphe en lignes.
2) En assumant que toutes les lignes sont tolérablement mauvaise, TEX utilise la médiocrité des lignes pour évaluer les différents moyens de couper le paragraphe en lignes. Durant cette évaluation il associe des "démérites" avec chaque ligne potentielle. La médiocrité augmente le nombre de démérites. TEX alors coupe le paragraphe en lignes d'une façon qui minimise les le total des démérites pour le paragraphe. Plus souvent TEX arrange le paragraphe pour minimiser la médiocrité de la ligne la pire. Voir pages 97–98 de *The TEXbook* et 99–99 de la traduction française pour les détails de comment TEX coupe un paragraphe en lignes.

La procédure de TEX pour assembler une séquence de lignes et autres matière en modes verticaux en pages est similaire à sa procédure de coupure de ligne. De toutes façons, assembler des pages n'est pas aussi compliqué parce que TEX ne considère qu'une page à la fois quand il cherche des coupures de page. Donc la seule décision qu'il doit prendre est où finir la page courante. De plus, quand TEX choisi des coupures de ligne il en considère plusieurs simultanément. (La plupart des traitements de texte choisissent des coupures de ligne une à la fois, et donc ne font pas un aussi bon boulot que celui que fait TEX) Voir pages 111–113 de *The TEXbook* et 999–999 de la traduction française pour les détails sur comment TEX choisi ses coupures de page.

mode. Quand TEX digère votre entrée dans son estomac (voir "Anatomie de TEX", p. 48), il est dans un des six *modes* :

- mode horizontal ordinaire (assemblage d'un paragraphe)
- mode horizontal restreint (assemblage d'une hbox)
- mode vertical ordinaire (assemblage d'une page)
- mode vertical restreint (assemblage d'une vbox)
- mode mathématique de texte (assemblage d'une formule apparaissant dans du texte)

- mode mathématique hors-texte (assemblage d'une formule apparaissant sur une ligne seule)

Le mode décrit une sorte d'entité que TEX rassemble.

Parce que vous pouvez encastrer une sorte d'entité avec une autre, c'est-à-dire, un vbox dans une formule, TEX garde trace non seulement d'un mode mais d'une liste entière de modes (ce qu'un informaticien appelle une "pile"). Supposez que TEX soit en mode M et rencontre quelque chose qui le met dans une nouveau mode M'. Quand il termine son travail dans le mode M', il reprend ce qu'il faisait dans le mode M.

mode horizontal. Quand TEX assemble un paragraphe ou une hbox, il est dans un des deux *modes horizontaux* : le mode horizontal ordinaire pour assembler des paragraphes et le mode horizontal restreint pour assembler des boîtes horizontales. A chaque fois que TEX est dans un mode horizontal son estomac (voir "Anatomie de TEX", p. 48) construit une liste horizontale d'articles (boîtes, ressort, pénalités, etc.). TEX compose les articles dans la liste l'un après l'autre, de gauche à droite.

Une liste horizontale ne peut contenir aucun articles produit par des commandes verticales internes, par exemple, `\vskip`.

- Si TEX assemble une liste horizontal dans le mode horizontal ordinaire et rencontre une commande verticale interne, TEX termine le paragraphe et entre dans le mode vertical.
- Si TEX assemble une liste horizontale dans le mode horizontal interne et rencontre une commande verticale interne, il rouspète.

Deux commandes que vous pensez d'abord horizontales internes sont en fait verticales internes : `\halign` (p. 184) et `\hrule` (p. 178). voir la page 286 de *The TEXbook* et 999 de la traduction française pour une liste des commandes verticales internes.

Vous devez faire attention à une subtile mais importante propriété du mode horizontal interne : *vous ne pouvez pas entrer en mode horizontal interne quand vous êtes en mode horizontal normal* Ce que cela signifie en pratique est que quand TEX assemble une hbox il n'appréhendera pas un texte en paragraphe, c'est-à-dire, du texte pour lequel il ferait une coupure de ligne. Vous pouvez contourner cette restriction en englobant le texte en paragraphe dans une vbox à l'intérieur de l'hbox. la même méthode marche si vous voulez mettre, disons un alignement horizontal dans un hbox.

mode mathématique. Un *mode mathématique* est un mode dans lequel est TEX quand il construit une formule mathématique. TEX a deux modes mathématiques différent : Le mode mathématique de texte pour construire une formule devant être englobée dans une ligne de texte, et le mode mathématique d'affichage pour construire une formule devant apparaître seule sur une ligne. Vous indiquez le mode mathématique de texte en englobant la formule entre `$` et dans le mode mathématique d'affichage en englobant la formule entre `$$`. Une propriété importante des deux

modes mathématiques est que les *espace saisis ne compte pas*. Voir les pages 290–293 de *The TEXbook* et 999–999 de la traduction française pour des détails sur la façon dont TEX réponds aux différentes commandes en mode mathématique.

mode ordinaire. Un *mode ordinaire* est un mode dans lequel est TEX quand il assemble un paragraphe en lignes ou assemble des lignes en une page. Voir "mode horizontal" (p. 81), "mode vertical" (p. 82).

mode restreint. Un *mode restreint* est un mode dans lequel est TEX quand il assemble une hbox ou une vbox. Nous suivons *The TEXbook* dans l'utilisation du terme "mode vertical interne" pour ce que vous vous attendez être un "mode vertical restreint". Voir "mode horizontal" (p. 81) et "mode vertical" (p. 82).

mode vertical. Quand TEX assemble soit une vbox soit la liste verticale principale dont les pages sont dérivées, il est dans un des deux *modes verticaux* : Le mode vertical ordinaire pour assembler la liste verticale principale et le mode vertical restreint pour assembler les vbox. à chaque fois que TEX est en mode vertical son estomac (voir "Anatomie de TEX", p. 48) construit une liste verticale d'éléments (boîtes, ressort, pénalités, etc.). TEX compose les éléments de la liste l'un après l'autre, de haut en bas.

Une liste verticale ne peut pas contenir tous les éléments produit par des commandes fondamentalement horizontales, c'est-à-dire, `\hskip` ou un caractère ordinaire (pas un espace) [13].

- Si TEX assemble une liste verticale dans le mode vertical ordinaire et rencontre une commande fondamentalement horizontale, il bascule vers le mode horizontal ordinaire.
- Si TEX assemble une liste verticale dans le mode vertical interne et rencontre une commande fondamentalement horizontale, il rouspète.

Deux commandes auxquelles vous devez penser en premier qu'elles sont fondamentalement verticale sont en fait fondamentalement horizontale : `\valign` (p. 185) et `\vrule` (p. 178). Voir la page 283 de *The TEXbook* et 999 de la traduction française pour une liste des commandes fondamentalement horizontales.

Il est particulièrement important d'avoir conscience que TEX considère un caractère ordinaire autre qu'un espace comme étant fondamentalement horizontal Si TEX débute soudainement un nouveau paragraphe quand vous ne vous y attendez pas une cause vraisemblable est un caractère que TEX a rencontré en étant en mode vertical. Vous pouvez convaincre TEX de ne pas traiter ce caractère comme fondamentalement horizontal en l'englobant dans une hbox puisque la commande `\hbox`, malgré son nom, n'est ni fondamentalement horizontale ni fondamentalement verticale.

[13] TEX *ignore* tout caractère espace qu'il rencontre quand il est dans un mode vertical.

mot de contrôle. Un *mot de contrôle* est une séquence de contrôle qui est constitué d'un caractère d'échappement suivi d'une ou plusieurs lettres[14]. TEX ignore tout espace ou fin de ligne qui suit un mot de contrôle, sauf pour noter qu'ils terminent le mot de contrôle.

muglue. Une *muglue* est une sorte de ressort que vous ne pouvez utiliser que dans les formules mathématiques. Elle est mesurée en `mu` (unités mathématiques). Une `mu` est égale à $^1/_{18}$ em, où la taille d'un em est prise de la famille 2 des polices mathématiques. TEX ajuste automatiquement la taille de la muglue en fonction du contexte. Par exemple, une taille de ressort de `2mu` est normalement plus petite dans un sous-script que dans un texte ordinaire. Vous devez utiliser la commande `\mskip` pour produire une muglue. Par exemple, '`\mskip 4mu plus 5mu`' produit un ressort mathématique avec un espace naturel de quatre `mu` et un étirement de cinq `mu`.

nom de fichier. Un *nom de fichier* nomme un fichier qui est connu du système d'exploitation qui à son tour supervise votre exécution de TEX. La syntaxe d'un nom de fichier ne doit *pas* suivre les règles usuelles de la syntaxe de TEX et en fait est différente selon les différentes implémentations de TEX. En particulier, la plupart des implémentations de TEX considère un nom de fichier comme étant terminé par un blanc ou une fin de ligne. Ainsi TEX est susceptible de mal interpréter '`{\input chapter2}`' en prenant l'accolade droite comme partie du nom de fichier. Comme règle générale, vous devez faire suivre un nom de fichier d'un blanc ou d'une fin de ligne comma dans '`{\input chapter2␣}`'.

nombre. En TEX, un *nombre* est un entier positif ou négatif. Vous pouvez écrire un nombre en TEX de quatre façons différentes :

1) comme un entier décimal ordinaire, par exemple, `52`
2) comme un nombre octal, par exemple, `'14`
3) comme un nombre hexadécimal, par exemple, `"FF0`
4) comme le code d'un caractère ASCII, par exemple, `` `) `` ou `` `\ ``

Chacune de ces formes peut être précédée par '`+`' ou '`-`'.

Un nombre octal ne peut avoir que les chiffres `0`–`7`. Un nombre hexadécimal peut avoir les chiffres `0`–`9` et `A`–`F`, représentant les valeurs de 0 à 15. Vous ne pouvez, hélas, utiliser de lettres minuscules quand vous écrivez un nombre hexadécimal. Si vous voulez des explications sur les nombres octaux et hexadécimaux, vous en trouverez dans les pages 43–44 de *The TEXbook* et 99–99 de la traduction française.

Un nombre décimal, octal ou hexadécimal se termine au premier caractère qui ne peut faire partie du nombre. Ainsi un nombre décimal se termine quand TEX voit, disons, une lettre, tandis qu'une lettre entre '`A`'

[14] Une "lettre" ici a la strict signification d'un caractère de code de catégorie 11.

et '`F`' ne terminera pas un nombre hexadécimal. Vous pouvez terminer un nombre avec un ou plusieurs espaces et normalement, TEX les ignorera[15].

La quatrième forme ci-dessus spécifie un nombre comme le code ASCII d'un caractère. TEX ignore les espaces après cette forme de nombre aussi. Vous pouvez écrire un nombre de cette forme soit comme '*c* soit comme '*c*. La seconde forme, bien que plus longue, a l'avantage que vous pouvez l'utiliser avec *tout* caractère, même '\', '`%`', ou '`^^M`'. En revanche, il y a un inconvénient plutôt technique : quand TEX développe une séquence de tokens pour une commande telle que `\edef` ou `\write`, des occurrences de '*c*' représentant un nombre sera aussi développée si elle peut l'être. C'est rarement l'effet que vous désirez.

Ce qui suit sont des représentations valides du nombre décimal 78 :

```
78   +078   "4E   '116   `N   `\N
```

Vous ne pouvez utiliser un nombre dans du texte par lui-même car un nombre n'est pas une commande. Néanmoins, vous pouvez insérer la forme décimal d'un nombre dans du texte en mettant une commande `\number` (p. 232) devant lui ou la forme numérique romaine en mettant une commande `\romannumeral` devant lui.

Vous pouvez aussi utiliser des constantes décimales, par exemple, des nombres avec une partie fractionnaire, pour spécifier des dimensions (voir "dimension", p. 61). Une constante décimale a un point décimal, qui peut être le premier caractère de la constante. Vous pouvez utiliser une virgule au lieu d'un point pour représenter le point décimal. Une constante décimale peut être précédée par un signe plus ou moins. Ainsi '`.5in`', '`-3.22pt`' et '`+1,5\baselineskip`' sont des dimensions valides. Vous ne pouvez, par contre, utiliser de constantes décimales dans tous les contextes *autre* que comme la partie "facteur" d'une dimension, c'est-à-dire, son multiplicateur.

outer. Une macro *outer* est une macro que vous ne pouvez utiliser dans certains contextes où TEX assemble des tokens à grande vitesse. Le but de rendre une commande outer est d'obliger TEX à générer des erreurs avant d'aller trop loin. Quand vous définissez une macro, vous pouvez la rendre outer avec la commande `\outer` (p. 239).

Vous ne pouvez utiliser de macro outer dans aucun des contextes suivants :

- comme argument d'une macro
- dans le texte paramètre ou le texte de remplacement d'une définition
- dans le préambule d'un alignement
- dans la partie non exécutée d'un test conditionnel

Un contexte outer est un contexte dans lequel vous pouvez utiliser une macro outer, c'est-à-dire c'est tout contexte autre que ceux listés ci-dessus.

[15] Quand vous définissez une macro qui se termine par un nombre, vous devez toujours mettre un espace après ce nombre ; autrement, TEX pourrait combiner ce nombre avec autre chose.

Par exemple, l'entrée suivante sera une utilisation interdite d'une macro outer

```
\leftline{\proclaim Assertion 2. That which is not inner
   is outer.}
```

La macro `\proclaim` (p. 136) est définie en plain TEX comme étant outer, mais elle est utilisée ici comme un macro argument à `\leftline`.

page. TEX compile un document en assemblant des *pages* une par une et en les passant à la routine de sortie. Quand il lit votre document, TEX maintient une liste des lignes et d'autres éléments à placer sur la page. (Les lignes sont normalement des hbox.) Cette liste est appelée la "liste verticale principale". Périodiquement TEX rentre dans un processus appelé "devoir du constructeur de page". Les éléments ajoutés à la liste verticale principale entre les devoirs du constructeur de page sont appelé "contributions récentes".

Le constructeur de page examine d'abord la liste verticale principale pour voir si elle a besoin de déposer une page maintenant, soit parce que les éléments sur la liste verticale principale ne tiennent pas sur la page, soit parce que un élément spécifique, tel que `\eject` (p. 143), demande à TEX de finir la page. S'il n'est pas nécessaire de déposer une page, alors le constructeur de page est fait pour la prochaine fois.

Autrement, le constructeur de page analyse la liste verticale principale pour trouver ce qu'il considère comme étant la meilleure coupure de page possible. Il associe pénalités avec plusieurs types de coupure de page peu attirante—une coupure qui laisse une ligne isolée en haut ou en bas de la page, une coupure juste avant un affichage mathématique, et ainsi de suite. Il choisit alors la coupure de page la moins chère où le coût d'une coupure est augmenté de toute pénalité associée avec cette coupure et par la médiocrité de la page qui en résulte (voir la page 111 de *The TEXbook* et 999 de la traduction française pour la formule du coût). S'il trouve plusieurs coupures de page de coût identique, il choisit la dernière.

Une fois que le constructeur de page a choisi une coupure de page, il place les éléments de la liste qui sont avant cette coupure dans la `\box255` et garde le reste pour la page suivante. Il appelle alors la routine de sortie. `\box255` agit comme une boîte aux lettres, avec le constructeur de page comme expéditeur et la routine de sortie comme destinataire. Ordinairement la routine de sortie traite la `\box255`, ajoute d'autres éléments, comme des insertions, entête et pieds de page à la page et envoi la page vers le fichier `.dvi` avec une commande `\shipout`. (Des routines de sortie spécialisées peuvent agir différemment.) Du point de vue de TEX, il importe peu que la routine de sortie envoie la page ou non ; La seule responsabilité de la routine de sortie est de traiter la `\box255` d'une façon ou d'une autre.

Il est important de réaliser que le meilleur endroit pour couper une page n'est pas nécessairement le dernier endroit possible de couper la page. Des pénalités et autres considérations peuvent faire que la coupure de page ait lieue plus tôt. De plus, TEX appréhende les éléments de la liste verticale

principale en batch, pas seulement un par un. Les lignes d'un paragraphe sont un exemple d'un tel batch. Pour ces raisons, le constructeur de page garde des éléments sous le coude quand il coupe une page. Ces éléments mis en réserve forment alors le début de la liste verticale principale de la page suivante (vraisemblablement au milieu d'un batch). Parce que des éléments sont déplacés d'un page à une autre, vous pouvez être sûr comme TEX exécute l'entrée, que le numéro de la page courante reflète la page sur laquelle la sortie correspondante apparaîtra. Voir les pages 110–114 de *The TEXbook* et 999–999 de la traduction française pour une description complète des règles de coupure de page de TEX.

paragraphe. Intuitivement, un *paragraphe* est une suite de ligne de saisie se terminant par une ligne blanche, une commande `\par` (p. 116) ou par une commande intrinsèquement verticale, telle que `\vskip`. Plus précisément, un paragraphe est une séquence de commandes que TEX exécute en mode horizontal restreint. Quand TEX a collecté un paragraphe entier, il le transforme en une séquence de lignes en choisissant les coupures de ligne (voir "coupure de ligne", p. 59). Le résultat est une liste de hbox avec ressort, pénalité interligne et matériel vertical imbriqué entre. Chaque hbox est une simple ligne et le ressort est le ressort inter-ligne.

TEX débute un paragraphe quand il est dans un mode vertical et rencontre une commande fondamentalement horizontal. En particulier, il est en mode vertical juste après avoir termine un paragraphe, donc le matériel horizontal sur la ligne suivant une ligne blanche débute le paragraphe suivant de manière naturelle. Il y a plusieurs types de commande fondamentalement horizontale, mais le type le plus commun est un caractère ordinaire, c'est-à-dire, une lettre.

Les commandes `\indent` et `\noindent` (pp. 117, 118) sont aussi des commandes fondamentalement horizontales qui demandent à TEX d'indenter ou non le début d'un paragraphe. Tout autre commande horizontal en mode vertical font que TEX fait un `\indent` implicite. Une fois que TEX a commencé un paragraphe, il est en mode horizontal ordinaire. Il commence par suivre toutes commandes situées dans `\everypar`. Il commence alors à collecter des éléments pour le paragraphe jusqu'a ce qu'il reçoive un signal de la fin du paragraphe. à la fin du paragraphe il réinitialise les paramètres de formation de paragraphe `\parshape`, `\hangindent`, et `\looseness`.

TEX traduit normalement une ligne blanche par `\par`. Il insère aussi un `\par` dans l'entrée chaque fois, en mode horizontal, qu'il rencontre une commande intrinsèquement verticale. Donc finalement la chose qui termine un paragraphe est toujours une commande `\par`.

Quand TEX reçoit une commande `\par`, il complète d'abord[16] le paragraphe sur lequel il travaille. Il coupe alors le paragraphe en lignes, ajoute la liste résultante d'éléments dans la liste verticale englobante et exécute le constructeur de page (dans le cas où la liste vertical englobante est la liste vertical principale). Si le paragraphe se termine par une commande intrinsèquement verticale, TEX exécute alors cette commande.

paramètre. Le terme *paramètre* a deux signification différentes—il peut faire référence soit à un paramètre de TEX, soit à un paramètre de macro.

Un paramètre de TEX est une séquence de contrôle qui nomme une valeur. La valeur d'un paramètre peut être un nombre, une dimension, un montant de ressort ou de muglue ou une liste de tokens. Par exemple, le paramètre `\parindent` spécifie la distance que TEX saute au début d'un paragraphe indenté.

Vous pouvez utiliser la séquence de contrôle d'un paramètre soit pour rétablir la valeur du paramètre soit pour fixer cette valeur. TEX interprète la séquence de contrôle comme une requête pour une valeur si elle apparaît dans un contexte où une valeur est attendue, et comme un assignement autrement. Par exemple :

```
\hskip\parindent
```

produit un ressort horizontal dont la taille naturelle est donnée par `\parindent`, tandis que :

```
\parindent = 2pc  % (ou \parindent 2pc)
```

fixe `\parindent` à une longueur de deux picas. L'assignement :

```
\parindent = 1.5\parindent
```

utilise `\parindent` des deux façons. Son effet est de multiplier la valeur de `\parindent` par 1.5.

Vous pouvez penser à un paramètre comme à un registre construit. Vous trouverez la liste de tous les paramètres de TEX dans les pages 272–275 de *The TEXbook* et 999–999 de la traduction française.

Un paramètre de macro est un endroit qui garde la place du texte qui est branché dans la définition d'une macro. Voir "macro" (p. 73) pour plus d'informations sur ce type de paramètre.

pénalité. Une *pénalité* est un élément que vous pouvez inclure dans une liste horizontale, verticale ou mathématique pour décourager TEX de couper la liste à cet endroit ou encourage TEX de couper la liste ici. Une pénalité positive indique un mauvais point de coupure, tandis qu'une pénalité négative indique un bon point de coupure. Couper une liste horizontal ordinaire produit une coupure de ligne, tandis que couper une

[16] Plus précisément, il exécute les commandes :

```
\unskip \penalty10000 \hskip\parfillskip
```

apportant ainsi les éléments de ces commandes à la fin de la liste horizontale courante.

liste verticale ordinaire produit une coupure de page. (Une pénalité n'a aucun effet en mode horizontal restreint ou vertical interne.)

Vous pouvez utiliser la commande `\penalty` (pp. 127, 142) pour insérer une pénalité explicitement. Une pénalité de 10000 ou plus empêche une coupure, tandis qu'une pénalité de −10000 ou moins force une coupure.

pied de page. Un *pied de page* est du matériel que TEX met en bas de chaque page, sous le texte de cette page. Le pied de page par défaut en plain TEX est un numéro de page centré. D'habitude un pied de page consiste en une ligne simple, que vous pouvez définir en assignant une liste de tokens à `\footline` (p. 149). Voir page 282 pour une méthode de production de pied de page multi-ligne.

plain TEX. *Plain TEX* est le format de TEX décrit dans ce livre et dans *The TEXbook*. Plain TEX est une partie du système TEX standard, donc des documents qui n'utilise que les aménagements de plain TEX peuvent facilement être transféré d'une installation vers une autre sans difficulté.

Plain TEX est constitué des commandes primitive auxquelles se joignent une large collection de macros et autres définitions. Ces définitions additionnelles sont données dans l'Annexe B de *The TEXbook* et de la traduction française. Elles doivent aussi être dans le fichier `plain.tex` quelque part sur votre système informatique.

point de référence. Le *point de référence* d'une boîte est le point où le coté gauche de la boîte est en intersection avec sa ligne de base. Quand TEX exécute une liste horizontale ou verticale, il utilise les points de référence des boîtes dans la liste pour aligner ces boîtes horizontalement ou verticalement (voir "boîtes", p. 51).

police. Une *police* en TEX est une collection de jusqu'a 256 caractères de sortie, ayant normalement les mêmes design, style (romain, italique, gras, condensé, etc.), et taille[17]. La police Computer Modern qui est fournie généralement avec TEX n'a que 128 caractères. Le colophon sur la dernière page de ce livre décrit les polices que nous avons utilisées pour composer ce livre.

Par exemple, voici l'alphabet en police romaine Palatino de 10 points :

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz

et le voici dans la police Computer Modern Grasse étendue de 12 point :

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz

[17] Plain TEX utilise une police spéciale pour construire les symboles mathématiques pour lesquelles les caractères ont différentes tailles. D'autres polices spéciales sont souvent utiles pour des applications comme le composition de logos.

Les caractères d'une police sont numérotés. La numérotation s'accorde normalement avec le numérotation ASCII pour les caractères existant dans le jeu de caractères ASCII. La table de code de chaque police indique à quoi ressemble le caractère avec le code n dans cette police. Certaines polices, comme celles utilisées pour les symboles mathématiques, n'ont pas de lettres. Vous pouvez produire une boîte contenant le caractère numéroté n, composé dans la police courante, en écrivant '`\char` n' (p. 105).

Pour utiliser une police dans votre document, vous devez d'abord la nommer avec une séquence de contrôle et la charger. Alors vous pourrez la sélectionner en saisissant cette séquence de contrôle quand vous voulez l'utiliser. Plain TEX procure un certain nombre de polices déjà nommées et chargées.

Vous nommez et chargez une police comme une simple opération, avec une commande comme '`\font\twelvebf=cmbx12`'. Ici, '`\twelvebf`' est la séquence de contrôle que vous utilisez pour nommer la police et '`cmbx12`' identifie le fichier de métrique de la police `cmbx12.tfm` sur votre système de fichier. Vous pouvez alors commencer à utiliser la police en saisissant '`\twelvebf`'. Après cela, la police sera effective jusqu'a ce que soit (a) vous sélectionner une autre police, soit (b) vous terminez le groupe, au cas où, dans lequel vous avez commencé avec la police. Par exemple, la saisie :

```
{\twelvebf white rabbits like carrots}
```

fera que la police `cmbx12` ne sera effective que pour le texte '`white rabbits like carrots`'.

Vous pouvez utiliser TEX avec d'autres police que Computer Modern (regardez l'exemple sur page 36 et les entêtes de page). Les fichiers de telles polices doivent être installés sur votre système de fichier à une place où TEX puisse les trouver. TEX et ses programmes compagnons ont généralement besoin de deux fichiers pour chaque police : un pour les métriques (`cmbx12.tfm`, par exemple) et un autre pour la forme des caractères (`cmbx12.pk`, par exemple). TEX lui-même n'utilise que les fichiers de métriques. Un autre programme, le pilote de périphérique, converti le fichier `.dvi` produit par TEX en une forme que votre imprimante ou autre périphérique de sortie puisse interpréter. Le pilote utilise le fichier de forme (s'il existe).

Le fichier de métrique de la police contient l'information dont TEX à besoin pour allouer de l'espace pour chaque caractère composé. Ainsi il inclue la taille de chaque caractère, les ligatures et crénages qui modifie les suites de caractères adjacent, et ainsi de suite. Ce que le fichier de métrique n'inclue *pas* sont les informations sur les formes des caractères de la police.

Le fichier de forme (pixel) peut être de différents formats. la partie d'extension du nom (la partie après le point) informe le pilote sur le format dans lequel est le fichier de forme. Par exemple, `cmbx12.pk` doit être le fichier de forme de la police `cmbx12` en format compressé, tandis que `cmbx12.gf` semble être le fichier de forme pour la police `cmbx12` en format

de police générique. Un fichier de forme peut ne pas être nécessaire pour une police résidente dans votre périphérique de sortie.

primitive. Une commande *primitive* est une commande dont la définition est construite dans le programme informatique TEX. Au contraire, une commande qui n'est pas une primitive est définie par une macro ou un autre format de définition écrit en TEX. Les commandes dans plain TEX sont constituée de commandes primitives auxquelles se joignent d'autres commandes définies en termes de primitives.

profondeur. La *profondeur* d'une boîte est la distance par laquelle la boîte s'étends sous sa ligne de base.

registre. Un *registre* est une place nommée pour stocker une valeur. C'est plus comme une variable dans un langage de programmation. TEX a cinq sortes de registres, comme montré dans le tableau suivant :

Type de registre	*Contenus*
`box`	une boîte
`count`	une nombre
`dimen`	une dimension
`muskip`	muglue
`skip`	ressort
`toks`	un token list

Les registres de chaque type sont numérotés de 0 à 255. Vous pouvez accéder au registre n de catégorie c en utilisant la forme '\cn', c'est-à-dire, `\muskip192`. Vous pouvez utiliser un registre n'importe où cette information de type approprié est appelée. Par exemple, vous pouvez utiliser `\count12` dans tout contexte appelant un nombre ou `\skip0` dans tout contexte appelant un ressort.

Vous mettez de l'information dans un registre en y assignant quelque chose :

```
\setbox3 = \hbox{lagomorphs are not mesomorphs}
\count255 = -1
```

Le premier assignement construit une hbox et l'assigne au registre de boîte 3. Vous pouvez ensuite utiliser '`\box3`' partout où une boîte est appelée et vous obtiendrez cette hbox[18]. Le second assignement assigne -1 au registre de compteur 255.

Un registre d'un type donné, par exemple, un registre de ressort, réagit comme un paramètre de ce type. Vous retrouvez sa value ou la lui assignez comme vous le feriez avec un paramètre. Quelques paramètres de TEX, par exemple, `\pageno`, sont implémentés comme des registres, en fait.

[18] Mais faites attention : utiliser un registre de boîte la vide aussi donc son contenu devient vide. Les autres sortes de registres n'ont pas cet manière de réagir. Vous pouvez utiliser la commande `\copy` (p. 170) pour conserver le contenu d'un registre de boîte sans le vider.

Plain TEX utilise plein de registres pour son propre usage, donc vous ne devez pas simplement prendre un numéro de registre arbitraire quand vous avez besoin d'un registre. à la place, vous devez demander à TEX de réserver un registre en utilisant une des commandes `\newbox`, `\newcount`, `\newdimen`, `\newmuskip`, `\newskip` ou `\newtoks` (p. 252). Ces commandes sont outer, donc vous ne pouvez pas les utiliser dans une définition de macro. Si vous le pouviez, vous utiliseriez un registre à chaque fois que la macro serait appelée et probablement dépasseriez le nombre de registres rapidement.

Quoiqu'il en soit, vous pouvez, avec précaution utiliser tout registre temporairement dans un groupe, même un que TEX utilise pour autre chose. Quand TEX a fini d'exécuter les commandes dans un groupe, il retrouve le contenu de tous les registres dans l'état où ils étaient avant de commencer à exécuter le groupe. Quand vous utilisez un registre numéroté explicitement dans un groupe, vous devez être sur que le registre n'est modifié par aucune macro que vous pourriez appelé dans le groupe. Soyez spécialement attentif quand vous utilisez des registres arbitraires dans un groupe qui appelle des macros que vous n'avez pas écrit vous-même.

TEX réserve certains registres à des tâches spéciales : de `\count0` à `\count9` pour des informations de numérotation de page et `\box255` pour le contenu d'une page juste avant qu'elle soit offerte à la routine de sortie. Les registres `\dimen0`–`\dimen9`, `\skip0`–`\skip9`, `\muskip0`–`\muskip9` et `\box0`–`\box9` ainsi que les `255` registres de boîte autres que `\box255` sont généralement utilisable comme registre "brouillon". Aussi plain TEX ne procure qu'un registre brouillon, `\count255`, pour des compteurs. Voir les pages 122 et 346 de *The TEXbook* et 999 et 999 de la traduction française pour des conventions à suivre pour choisir des numéros de registre.

Vous pouvez examiner le contenu des registres lors d'une exécution avec la commande `\showthe` (p. 261), par exemple, '`\showthe\dimen0`'.

réglures. Vous pouvez utiliser des réglures pour remplir un espace avec des copies d'un modèle, c'est-à-dire, mettre des points répétés entre un titre et un numéro de page dans une table des matières. une réglure est une copie simple d'un modèle. La spécification des réglures contient trois parties d'information.

1) ce qu'est une réglure seule
2) combien d'espace doit être rempli
3) comment les copies du modèle doivent être arrangée dans l'espace

TEX possède trois commandes pour spécifier des réglures : `\leaders`, `\cleaders` et `\xleaders` (p. 179). L'argument de chaque commande spécifie la réglure. La commande doit être suivie d'un ressort ; La taille du ressort spécifie combien d'espace doit être rempli. Le choix de la commande détermine comment les réglures sont arrangée dans l'espace.

Voici un exemple montrant comment les `\leaders` travaillent :

```
\def\dotting{\leaders\hbox to 1em{\hfil.\hfil}\hfil}
```

```
\line{The Political Process\dotting 18}
\line{Bail Bonds\dotting 26}
```

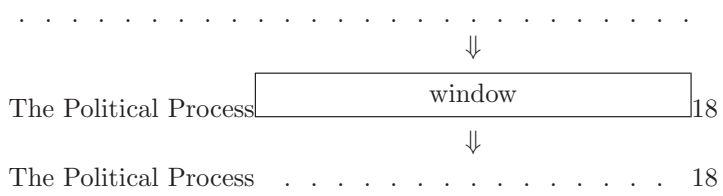
Ici, nous avons placé les réglures et leur ressort associé dans une définition de macro pour pouvoir les utiliser facilement à deux endroit. Cette entrée produit :

The Political Process 18
Bail Bonds 26

La hbox suivant `\leaders` spécifie la réglure, soit, une hbox d'1 em de large contenant un point en son centre. L'espace est rempli avec des copies de cette boîte, effectivement remplies de points dont les centres sont séparés de 1 em. Le `\hfil` suivant (celui à la fin de la définition de macro) est un ressort qui spécifie l'espace à remplir. Dans ce cas, c'est l'espace nécessaire pour remplir la ligne. en choisissant `\leaders` plutôt que `\cleaders` ou `\xleaders`, nous nous assurons que les points des différentes lignes soient alignés entre eux.

En général, l'espace à remplir agit comme une fenêtre pour les copies répétées de réglures. TEX insère autant de copies que possible, mais de l'espace est habituellement laissé—soit parce que les réglures s'arrêtent dans la fenêtre, soit parce que la largeur de la fenêtre n'est pas un multiple exacte de la largeur de la réglure. La différence entre les trois commandes est dans leur façon d'arranger les réglures dans la fenêtre et comment elle distribuent l'espace perdu.

- Pour `\leaders`, TEX produit d'abord une rangée de copies de la réglure. Il aligne alors le début de cette rangée avec l'extrémité gauche de la boîte la plus interne B qui contient le résultat de la commande `\leaders`. Dans les deux lignes d'exemple ci-dessous, B est une boîte produite par `\line`. Ces réglures qui rentrent entièrement dans la fenêtre sont placées dans B, et l'espace restant aux extrémités gauche et droite est laissé vide. L'image est comme ceci :



  Cette procédure assure que dans les deux lignes d'exemple de la page précédente, les points des deux ligne soient alignés verticalement (puisque les points de référence des hbox produites par `\line` sont alignés verticalement).
- Pour `\cleaders`, TEX centre les réglures dans la fenêtre en divisant l'espace perdu entre les deux extrémités de la fenêtre. L'espace perdu est toujours inférieur à la largeur d'une seule réglure.
- Pour `\xleaders`, TEX distribue l'espace perdu de chaque coté dans la fenêtre. En d'autres mots, si l'espace perdu est w et que la réglure

se répète n fois, TEX met un espace de largeur $w/(n+1)$ entre les réglures adjacente et à deux extrémités des réglures. L'effet est habituellement d'étaler les réglures une petit peu. l'espace perdu pour `\xleaders`, comme celui de `\cleaders`, est toujours inférieur à la largeur d'une réglure seule.

Jusqu'ici, nous supposions que les réglures consistaient en hbox arrangées horizontalement. Deux variations sont possibles :

1) Vous pouvez utiliser une règle à la place d'une hbox pour la réglure. TEX rend la règle aussi large que possible pour s'étendre au delà du ressort (et les trois commandes sont équivalentes).
2) Vous pouvez produire des réglures verticales qui traversent la page en les incluant dans une liste verticale au lieu d'une liste horizontale. Dans ce cas vous avez besoin de ressort vertical après les réglures.

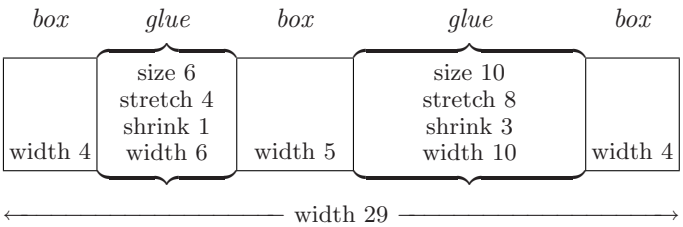
Voir les pages 223–225 de *The TEXbook* et 999–999 de la traduction française pour les règles précises que TEX utilise en composant des réglures.

ressort. Le *ressort* est de l'espace blanc qui peut se rétrécir ou s'étirer. Le ressort donne à TEX la flexibilité dont il a besoin pour produire des documents impeccables. Le ressort existe en deux types : le ressort horizontal et le ressort vertical. Le ressort horizontale apparaît dans des listes horizontales, tandis que le ressort verticale apparaît dans des listes verticales. Vous pouvez produire un ressort soit implicitement, c'est-à-dire, avec un espace inter-mot, soit explicitement, c'est-à-dire, avec la commande `\hskip`. TEX lui-même produit beaucoup de ressort en composant votre document. Nous ne décrirons que le ressort horizontal—le ressort vertical étant analogue.

Quand TEX assemble une liste de boîtes et de ressort en grande quantité, il ajuste la taille des ressorts pour garder les espaces demandées d'unité de largeur. Par exemple, TEX s'assure que la marge de droite d'une page est uniforme en ajustant les ressorts horizontaux des lignes. Similairement, il d'assure que les différentes pages ont la même marge du bas en ajustant les ressorts entre les blocs de texte comme des paragraphes et des affichages mathématiques (où le changement est probablement moins remarquable).

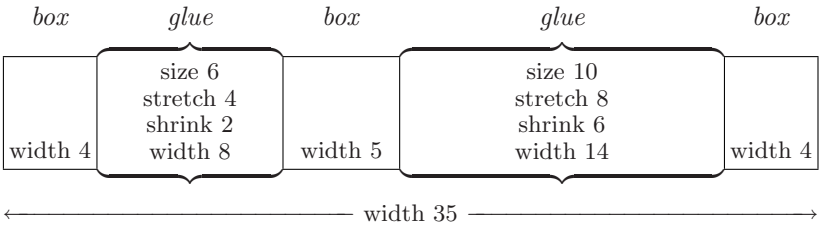
Un ressort a un espacement naturel—la taille qu'il "veut avoir". Le ressort a aussi deux autres attributs : son étirement et son rétrécissement. Vous pouvez produire un montant spécifique de ressort horizontal avec la commande `\hskip` (p. 161). La commande `\hskip 6pt plus 2pt minus 3pt` produit un ressort horizontal dont la taille naturelle est de 6 points, l'étirement de 2 points et le rétrécissement 3 points. Similairement, vous pouvez produire un montant spécifique de ressort vertical avec la commande `\vskip` commande (p. 161).

Le meilleur moyen de comprendre ce qui s'étire et ce qui se rétrécie et de voir un exemple de ressort au travail. Supposez que vous construisez un hbox de trois boîtes et deux ressorts, comme dans l'image :



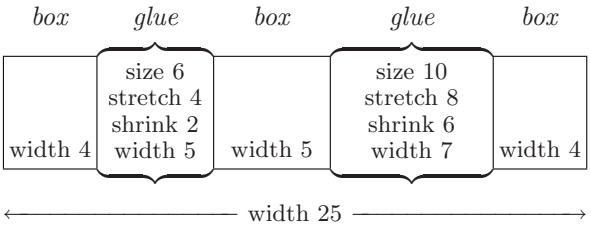
L'unité de mesure ici peut être en points, en millimètres, etc. Si le hbox est autorisé à assumer sa largeur naturelle, alors chaque ressort de la boîte assume aussi sa largeur naturelle. La largeur totale du hbox est alors la somme des largeurs de ses parties, soit, 29 unités.

Ensuite, supposez que le hbox doit être plus large que 29 unités, disons 35 unités. Cela peut arrivez, par exemple, si le hbox doit occuper une ligne entière et que la largeur de la ligne soit de 35 unités. Puisque les boîtes ne peuvent changer leurs largeur, TEX produit l'espace supplémentaire nécessaire en rendant les ressorts plus large. L'image maintenant ressemble à ceci :



Les ressorts ne deviennent pas plus large de manière égale ; elle deviennent plus large en proportion de leur étirement. Puisque le second ressort a deux fois plus d'étirement que le premier, il devient plus large de quatre unités tandis que le premier devient plus large de seulement deux unités. Le ressort peut s'étirer autant que nécessaire, néanmoins, TEX rechigne à l'étirer au delà du montant d'étirement donné dans sa définition.

Finalement, supposez que le hbox doive se rapprocher de 29 unités à, disons 25 unités. Dans ce cas TEX fait se rétrécir les ressorts. L'image ressemble à ceci :



Les ressorts deviennent plus proches en proportion de leur rétrécissement. Le premier ressort devient plus étroit d'une unité, tandis que le second rétrécit de trois unités. Le ressort ne peut pas se rétrécir d'un distance inférieure au montant de rétrécissement donné dans sa définition

même si la distance à laquelle il peut se rétrécir est sans limite. Sur ce point important l'étirement et le rétrécissement réagissent différemment.

Une bonne règle à suivre pour le ressort est de fixer sa taille normale au montant d'espace qui va le mieux, l'étirement au montant d'espace le plus grand que TEX puisse ajouter avant que le document devienne mauvais et le rétrécissement au montant d'espace le plus grand que TEX puisse enlever avant que le document commence à devenir mauvais. Vous pouvez devoir fixer les valeurs par expérimentation.

Vous pouvez produire un ressort étirable à l'infini en spécifiant son étirement en unités de '`fil`', '`fill`' ou '`filll`'. Le ressort mesuré en '`fill`' est infiniment plus étirable que le ressort mesuré en '`fil`' et le ressort mesuré en '`filll`' est infiniment plus étirable que le ressort mesuré en '`fill`'. Vous n'aurez que très rarement besoin de ressort '`filll`'. Un ressort s'étirant de `2fil` s'étire deux fois plus qu'un ressort s'étirant de `1fil`, et de même pour les autres sortes de ressort étirable infiniment.

Quand TEX apporte un espace supplémentaire en plus d'un ressort, celui infiniment étirable, s'il y en a, en prend la totalité. Le ressort infiniment étirable est particulièrement pratique pour cadrer du texte à gauche, à droite ou centré.

- Pour faire un texte cadré à gauche, mettez un ressort horizontal infiniment étirable à sa droite. Ce ressort consommera tout l'espace supplémentaire possible sur la ligne. Vous pouvez utiliser la commande `\leftline` (p. 115) ou la commande `\raggedright` (p. 122) pour faire cela.
- Pour faire un texte cadré à droite, mettez un ressort horizontal infiniment étirable à sa gauche. comme ci-dessus ce ressort consommera tout l'espace supplémentaire possible sur la ligne. Vous pouvez utiliser la commande `\rightline` (p. 115) pour faire cela.
- Pour faire du texte centré, mettez un ressort horizontal infiniment étirable de chaque coté. Ces deux ressort diviseront tout l'espace supplémentaire de la ligne également entre eux. Vous pouvez utiliser la commande `\centerline` (p. 115) pour faire cela.

Vous pouvez aussi spécifier un ressort rétrécissable infiniment de la même manière. Un ressort rétrécissable infiniment peut agir comme espace négatif. Notez que `fil`, etc., ne peuvent être utilisé que pour spécifier l'étirement et le rétrécissement du ressort—Ils ne peuvent être utilisé pour spécifier sa taille normale.

ressort inter-ligne. Le *ressort inter-ligne* est le ressort que TEX insère au début de chaque boîte dans une liste verticale sauf pour la première. Le ressort inter-ligne est normalement spécifiée de façon à maintenir une distance constante entre les lignes de base des boîtes. Sa valeur est déterminée conjointement par les paramètres `\baselineskip`, `\lineskip` et `\lineskiplimit` (p. 139).

rétrécissement. Voir "ressort" (p. 93).

routine de sortie. Quand TEX a accumulé au moins assez de matériel pour remplir une page, il choisi un point d'arrêt et place le matériel situé avant le point d'arrêt en `\box255`. Il appelle alors la *routine de sortie* courante, qui compile le matériel et l'envoie éventuellement dans le fichier `.dvi`. La routine de sortie peut faire d'autres actions, comme insérer des entêtes, des pieds de page et des notes de pieds de page. Plain TEX procure une routine de sortie par défaut qui insère un numéro de page centré en bas de chaque page. En procurant une routine de sortie différente vous pouvez créer des effets tels que des sortie en double colonnes. Vous pouvez imaginer les routines de sortie comme ayant une seule responsabilité : disposer le matériel en `\box255` d'une façon ou d'une autre.

La routine de sortie courante est définie par la valeur de `\output` (p. 154), qui est une liste de tokens. Quand TEX est prêt à produire une page, il développe simplement la liste de tokens.

Vous pouvez faire quelque simples changements aux actions de la routine de sortie de plain TEX sans la modifier réellement. Par exemple, en assignant une liste de tokens à `\headline` ou `\footline` (p. 149) vous pouvez faire que TEX produise un entête ou un pied de page différent de l'ordinaire.

La routine de sortie est aussi responsable de collecter toutesinsertions ; en combinant ces insertions et toutes "décorations" telles que entêtes et pieds de page avec les contenus principaux de la page et empaquète tout ce matériel dans une boîte ; et éventuellement envoyer cette boîte dans le fichier `.dvi` avec la commande `\shipout`(p. 154). Bien que cela soit ce que fait une routine de sortie le plus souvent, une routine de sortie d'usage spécial peut agir différemment.

séquence de contrôle. Une *Séquence de contrôle* est un nom pour une command TEX. Une control Séquence de contrôle commence par un caractère d'échappement, habituellement un antislash (\). Une Séquence de contrôle prend une des deux formes :

- Un mot de contrôle est une Séquence de contrôle constituée d'un caractère d'échappement suivi par une ou plusieurs lettres. Le mot de contrôle se termine quand TEX voit une non-lettre. Par exemple, quand TEX lit '`\hfill␣,␣the`', il voit six tokens : Les séquences de contrôle '`\hfill`', virgule, espace, '`t`', '`h`', '`e`'. L'espace après '`\hfill`' termine la séquence de contrôle et est absorbée par TEX quand il lit la séquence de contrôle. (Pour le texte '`\hfill,␣the`', d'un autre coté, La virgule termine aussi la séquence de contrôle et compte comme un caractère de bon aloi.)
- Un symbole de contrôle est une séquence de contrôle constituée d'une caractère d'échappement suivi par n'importe caractère autre qu'une lettre—même un espace ou une fin de ligne. Un symbole de contrôle est auto-délimité, c'est-à-dire, TEX sait où il termine sans devoir

regarder le caractère qui le suit. Le caractère après un symbole de contrôle n'est jamais absorbée par le symbole de contrôle.

Voir page 66 pour plus d'information sur les espaces après des séquences de contrôle.

TEX procure une grand nombre de séquences de contrôle prédéfinie. Les séquences de contrôle primitives sont construite dans le programme informatique TEX et donc, sont accessible sous toutes les formes de TEX. D'autres séquences de contrôle sont fournies par plain TEX, la forme de TEX décrite dans ce livre

Vous pouvez augmenter des séquences de contrôle prédéfinies avec celles de votre cru, en utilisant des commandes telles que `\def` et `\let` pour les définir. La section 12 de ce livre contient une collection de définitions de séquence de contrôle que vous pouvez trouver utile. De plus, votre facilité informatique peut vous rendre capable de créer une collection de macros TEX développées localement.

strut. Un *strut* est une boîte invisible dont la largeur est à zéro et dont la hauteur et la profondeur sont légèrement supérieure à celle d'une ligne de texte "normale" dans le contexte. Les struts sont pratique pour obtenir un espacement vertical uniforme quand l'espacement de ligne usuel de TEX, par exemple, dans une formule mathématique ou dans un alignement horizontal où vous spécifiez `\offinterlineskip`. Parce qu'un strut est plus haut et plus profond que tout autre chose sur sa ligne, il détermine la hauteur et la profondeur de la ligne. Vous pouvez produire un strut avec la commande `\strut` (p. 173) ou la commande `\mathstrut` (p. 174). Vous pouvez utiliser `\strut` n'importe où, mais vous ne pouvez utiliser `\mathstrut` que quand TEX est en mode mathématique. Un strut dans plain TEX a une hauteur de 8.5 pt et une profondeur de 3.5 pt, tandis qu'un strut mathématique à la hauteur et la profondeur d'une parenthèse ouvrante du style courant (donc il est plus petit pour des indices ou des exposants).

Voici un exemple montrant comment vous devriez utiliser un strut :

```
\vbox{\hsize = 3in \raggedright
   \strut Here is the first of two paragraphs that we're
   setting in a much narrower line length.\strut}
\vbox{\hsize = 3in \raggedright
   \strut Here is the second of two paragraphs that we're
   setting in a much narrower line length.\strut}
```

Cette entrée donne :

Here is the first of two paragraphs that we're setting in a much narrower line length.

Here is the second of two paragraphs that we're setting in a much narrower line length.

Sans les struts la vbox serait trop rapprochée. Similairement, dans la formule :

```
$\overline{x\mathstrut} \otimes \overline{t\mathstrut}$
```

Les struts mathématiques font que les deux barres sont mises à la même hauteur même si le 'x' et le 't' ont des hauteurs différentes :

$\overline{x} \otimes \overline{t}$

style. Le matériel d'une formule mathématique est mis dans un des huit *styles*, dépendant du contexte. Connaître les styles peut être utile si vous voulez mettre une partie d'une formule dans une taille de caractère différente de celle que TEX a choisi en fonction de ses règles usuelles.

Les quatre styles primaires sont :

display style	(for formules affichées sur une ligne seule)
text style	(for formules englobées dans du texte ordinaire)
script style	(for exposants et indices)
scriptscript style	(for indices d'indices, etc.)

Les quatre autres styles sont dit variantes étroites. Dans ces variantes les exposants ne sont pas montés aussi haut que normalement et donc la formule nécessite moins d'espace vertical qu'ils le devraient. Voir les pages 140–141 de *The TEXbook* et 999–999 de la traduction française pour les détails sur la façon de TEX de sélectionner le style.

TEX choisi une taille de caractère en fonction du style :

- Le style d'affichage et le style texte sont mis en taille texte, comme 'ceci'.
- le style script est mis en taille script, comme '$_{\text{ceci}}$'.
- le style scriptscript est mis en taille scriptscript, comme '$_{_{\text{ceci}}}$'.

Voir "famille" (p. 67) pour plus d'informations à propos de ces trois taille.

TEX n'a pas de style "scriptscriptscript" parce qu'un tel style serait souvent mis dans une taille de caractère trop petite à lire. TEX par conséquent met les indices, exposants, etc. de troisième ordre en utilisant le style scriptscript.

Parfois vous pourrez trouver que TEX a mis une formule dans un style différent de celui que vous préférez. Vous pouvez passer outre le choix de TEX avec les commandes `\textstyle`, `\displaystyle`, `\scriptstyle` et `\scriptscriptstyle` (p. 206).

symbole de contrôle. Un *symbole de contrôle* est une séquence de contrôle qui est constitué d'un caractère d'échappement suivi d'un caractère autre qu'une lettre—même un espace et une fin de ligne.

taille script. Une *taille script* décrit une des trois polices reliées dans une famille. Une taille script est plus petite que la taille texte mais plus grande que la taille scriptscript. TEX utilise la taille script pour les indices

et les exposants, aussi bine que pour les numérateurs et les dénominateurs de fractions dans du texte.

taille scriptscript. Une *taille scriptscript* décrit la plus petite taille des trois polices reliées dans une famille. TEX utilise la taille scriptscript pour des indices, exposants, numérateurs et dénominateurs de second rang. Par exemple, TEX utilisera la taille scriptscript pour un indice d'indice ou pour un exposant d'exposant.

taille texte. La *taille texte* décrit la grande des trois polices liées dans une famille. TEX utilise la taille texte pour des symbole ordinaire apparaissant en mode mathématique.

test conditionnel. Un *test conditionnel* est une commande qui teste si une condition est vraie ou non et demande à TEX soit de développer soit de sauter du texte, selon le cas. La forme générale d'une test conditionnel est soit :

`\if`α⟨*true text*⟩`\else`⟨*false text*⟩`\fi`

soit :

`\if`α⟨*true text*⟩`\fi`

où α spécifie le test particulier. Par exemple, `\ifvmode` teste la condition que TEX soit actuellement dans un mode vertical. Si la condition est vraie, TEX développe ⟨*true text*⟩. Si la condition est fausse, TEX développe ⟨*false text*⟩ (s'il est présent). Les tests conditionnel sont interprétés dans l'œsophagede TEX(voir "Anatomie de TEX", p. 48), donc tous les tokens développable dans le texte interprété ne sont développé qu'après que le test n'ait été résolu. Les différents tests conditionnels sont expliqué dans "Tests conditionnels" (p. 243).

TEX MEX. (a) Une variété de TEX utilisée pour composer des mathématiques dans les pays d'Amérique Centrale. (b) Une cuisine très épicée appréciée par les TEXniciens d'El Paso.

texte justifié. Du *texte justifié* est du texte qui a été composé pour que les deux marges soit alignées. Du texte non justifié, d'un autre coté, a été composé avec des marges "déchirées" d'un ou des deux cotés. Les documents saisis sur les vieilles machines à écrire ont la plupart du temps des marges droites déchirées. Bien que les documents produits par TEX soit justifié par défaut, vous pouvez si vous voulez produire des documents (ou des suites de lignes) qui ont la marge droite déchirée—ou gauche déchirée. Vous pouvez aussi demander à TEX de centrer un suite de lignes, ce qui rend les deux marges déchirées. Vous pouvez utiliser les commandes `\leftskip`, `\rightskip` et `\raggedright` (pp. 121, 122) pour cela.

Quand TEX produit du texte justifié, il a normalement besoin de rétrécir ou d'étirer les ressorts de chaque ligne pour que les marges soit

alignées. Quand TEX produit du texte non justifié, d'un autre coté, il laisse les ressorts de chaque ligne à leur taille naturelle. Beaucoup de typographes préfèrent le texte non justifié parce que son espacement inter-mot est plus uniforme.

texte mathématique. Nous utilisons le terme *texte mathématique* pour faire référence à une formule mathématique mise dans une ligne de texte, c'est-à-dire, entourée de `$`. TEX mets du texte mathématique dans le mode texte mathématique .

token. Un *token* est soit un simple caractère attaché à un code de catégorie ou une séquence de contrôle. TEX lit les caractères d'un fichier en utilisant ses yeux (voir "Anatomie de TEX", p. 48) et groupe les caractères en tokens en utilisant sa bouche. Quand un token atteint l'estomac de TEX, TEX l'interprète comme une commande à moins qu'il fasse partie d'un argument d'une commande précédente.

unité de mesure. Voir "dimension" (p. 61).

unité mathématique. Une *unité mathématique*, codée '`mu`', est une unité de distance utilisée pour spécifier un ressort dans les formules mathématiques. Voir le "muglue" (p. 83).

vbox. Une *vbox* (boîte verticale) est une boîte que TEX construit en plaçant les éléments d'une liste verticale l'une après l'autre, de haut en bas. Une vbox, prise en tant qu'unité, n'est ni fondamentalement horizontale ni fondamentalement verticale, c'est-à-dire, elle peut apparaître comme un élément soit d'une liste verticale soit d'une liste horizontale. Vous pouvez construire une vbox avec les commandes `\vbox` ou `\vtop` (p. 167). La différence est que pour `\vbox`, le point de référence de la vbox construite est dérivé de celui du dernier (et souvent plus bas) élément constituant la liste, mais pour `\vtop`, c'est celui du premier (et normalement plus haut) élément constituant la liste.

5 Commandes pour composer des paragraphes

Cette section couvre les commandes qui traitent des caractères, des mots, des lignes et des paragraphes entiers. Pour une explication des conventions utilisées dans cette section, voir les "Descriptions de commandes" (p. 3).

Caractères et accents

Lettres et ligatures pour alphabets Européens

`\AA` Lettre scandinave Å
`\aa` Lettre scandinave å
`\AE` Ligature Æ
`\ae` Ligature æ
`\L` Lettre Polonaise Ł
`\l` Lettre Polonaise ł
`\O` Lettre Danoise/Norvégienne Ø
`\o` Lettre Danoise/Norvégienne ø
`\OE` Ligature Œ
`\oe` Ligature œ
`\ss` Lettre Allemande ß

Ces commandes produisent diverses lettres et ligatures des alphabets européens. Elles sont utiles pour des mots et expressions occasionnels dans ces langues—mais si vous devez composer une grande quantité de texte dans une langue européenne, vous feriez probablement mieux d'utiliser une version de TEX adaptée à cette langue.[1]

[1] Le TEX Users Group (p. 18) peut vous fournir des informations sur les versions européennes de TEX.

Vous aurez besoin d'un espace après ces commandes quand vous les utiliserez dans un mot pour que TEX traite les lettres suivantes comme élément du mot et non en tant qu'élément de la commande. Vous n'avez pas besoin d'être en mode mathématique pour utiliser ces commandes.

Exemple :

```
{\it les \oe uvres de Moli\‘ere}
```

produit :

les œuvres de Molière

Symboles spéciaux

☞ `\#` signe dièse #
`\$` signe dollar $
`\%` signe pourcentage %
`\&` esperluette &
`\_` souligné _
`\lq` quote gauche ‘
`\rq` quote droite ’
`\lbrack` crochet gauche [
`\rbrack` crochet droit]
`\dag` symbole dague †
`\ddag` symbole double dague ‡
`\copyright` symbole copyright ©
`\P` symbole paragraphe ¶
`\S` symbole section §

Ces commandes produisent divers marques et caractères spéciaux. Les cinq premières commandes sont nécessaires parce que TEX attache par défaut des significations spéciales aux caractères (`#`, `$`, `%`, `&`, `_`). vous n'avez pas besoin d'être en mode mathématique pour employer ces commandes.

Vous pouvez utiliser le signe dollar de la police Computer Modern italique pour obtenir le symbole livre sterling, comme montré dans l'exemple ci-dessous.

Exemple :

```
\dag It’ll only cost you \$9.98 over here, but in England
it’s {\it \$}24.98.
```

produit :

†It'll only cost you $9.98 over here, but in England it's £24.98.

`\TeX`

Cette commande produit le logo TEX. souvenez vous de la faire suivre par \␣ ou de l'inclure dans un groupe quand vous voulez un espace après lui.

Exemple :

```
A book about \TeX\ is in your hands.
```

produit :

A book about TEX is in your hands.

`\dots`

Cette commande produit des points de suspension, c'est-à-dire, trois points, dans du texte ordinaire. Elle est prévue pour être utilisée en écriture mathématique ; pour des points de suspension entre des mots ordinaires, vous devez utiliser `$\ldot$` (p. 211) à la place. Puisque `\dots` inclut son propre espace, vous ne devez pas le faire suivre de \␣.

Exemple :

```
The sequence $x_1$, $x_2$, \dots, $x_\infty$
does not terminate.
```

produit :

The sequence x_1, x_2, …, x_∞ does not terminate.

Voir aussi : "Divers symboles mathématiques ordinaires" (p. 196).

Caractères Arbitraires

`\char` ⟨*charcode*⟩

Cette commande produit le caractère situé à la position ⟨*charcode*⟩ de la police courante.

Exemple :

```
{\char65} {\char `A} {\char `\A}
```

produit :

A A A

`\mathchar` ⟨*mathcode*⟩

Cette commande produit le caractère mathématique dont la classe, la famille et la position dans la police sont données par ⟨*mathcode*⟩. Elle n'est légale qu'en mode mathématique.

Exemple :

```
\def\digger{\mathchar "027F} % Like \spadesuit in plain TeX.
% Class 0, family 2, font position "7F.
$\digger$
```

produit :

♠

Voir aussi : `\delimiter` (p. 213).

Accents

`\'` accent aigu comme dans é
`\.` accent point comme dans ṅ
`\=` accent macron comme dans r̄
`\^` accent circonflexe comme dans â
`\‘` accent grave comme dans è
`\"` accent tréma comme dans ö
`\~` accent tilde comme dans ã
`\c` accent cédille comme dans ç
`\d` accent point en dessous comme dans ṛ
`\H` accent tréma Hongrois comme dans ő
`\t` accent tie-after comme dans u͡u
`\u` accent bref comme dans ř
`\v` accent circonflexe inversé comme dans ǒ

Ces commandes produisent des accents en texte ordinaire. Vous devez normalement laisser un espace après ceux notés par une simple lettre. (see “Espaces”, p. 12).

Exemple :

```
Add a soup\c con of \'elan to my pin\~a colada.
```

produit :

Add a soupçon of élan to my pinã colada.

`\i`
`\j`

Ces commandes produisent des versions sans point des lettres ‘i’ et ‘j’. Vous devez les utiliser au lieu des ‘i’ et ‘j’ ordinaires quand vous mettez un accent sur ces lettres dans du texte ordinaire. Utilisez les commandes `\imath` et `\jmath` (p. 196) pour des ‘i’ et ‘j’ sans point dans des formules mathématiques.

Exemple :

```
long ‘i’ as in l\=\i fe  \quad \v\j
```

produit :

long ‘i’ as in līfe ǰ

`\accent` ⟨*charcode*⟩

Cette commande met un accent au-dessus du caractère suivant cette commande. L'accent est le caractère de position ⟨*charcode*⟩ dans la police courante. TEX suppose que l'accent a été conçu pour s'adapter sur un caractère d'une hauteur de 1 ex dans la même police que l'accent. Si le caractère à accentuer est plus haut ou plus petit, TEX ajuste la position en conséquence. Vous pouvez changer de police entre l'accent et le caractère suivant, cela dessine le caractère accent et le caractère à accentuer venant

de différentes polices. Si le caractère accent n'est pas vraiment prévu pour être un accent, TEX ne se plaindra pas ; il composera juste quelque chose de ridicule.

Exemple :

```
l'H\accent94 otel des Invalides
% Position 94 of font cmr10 has a circumflex accent.
```

produit :

l'Hôtel des Invalides

Voir aussi : Accents mathématiques (p. 207).

Defeating boundary ligatures

`\noboundary`

Vous pouvez défaire une ligature ou un crénage que TEX applique au premier ou au dernier caractère d'un mot en mettant `\noboundary` juste avant ou après ce mot. Certaines polices prévues pour des langues autres que l'anglais contiennent un caractère spécial de frontière que TEX met au commencement et à la fin de chaque mot. Le caractère de frontière n'occupe aucun espace et est invisible une fois imprimé. Il permet à TEX de fournir un traitement typographique différent aux caractères de début ou de fin de mot, puisque le caractère de frontière peut faire partie d'une séquence de caractères devant être crénelée ou remplacée par une ligature. (aucunes polices standards de TEX ne contient ce caractère de frontière.) L'effet de `\noboundary` est de supprimer le caractère de frontière s'il est là, et de ce fait d'empêcher TEX d'identifier la ligature ou le crénage.

Sélectionner des polices

Polices particulières

`\fivebf` utilise une police grasse de 5 points
`\fivei` utilise une police mathématique italique de 5 points
`\fiverm` utilise une police romaine de 5 points
`\fivesy` utilise une police de symbole mathématique de 5 points
`\sevenbf` utilise une police grasse de 7 points
`\seveni` utilise une police mathématique italique de 7 points
`\sevenrm` utilise une police romaine de 7 points
`\sevensy` utilise une police de symbole mathématique de 7 points
`\tenbf` utilise une police grasse de texte de 10 points
`\tenex` utilise une police d'extension mathématique de 10 points
`\teni` utilise une police mathématique italique de 10 points
`\tenrm` utilise une police romaine de texte de 10 points
`\tensl` utilise une police romaine penchée de 10 points
`\tensy` utilise une police de symbole mathématique de 10 points
`\tenit` utilise une police italique de 10 points
`\tentt` utilise une police de type machine à écrire de 10 points

Le texte suivant ces commandes est composé par TEX dans la police indiquée. Normalement vous devriez enfermer une de ces commandes de choix de police dans un groupe avec le texte devant être composé dans la police choisie. En dehors d'un groupe une commande de choix de police est efficace jusqu'à la fin du document (à moins que vous ne l'annuliez avec une autre commande de ce genre).

Exemple :

```
See how I've reduced my weight---from
120 lbs.\ to {\sevenrm 140 lbs}.
```

produit :

See how I've reduced my weight—from 120 lbs. to 140 lbs.

`\nullfont`

Cette commande sélectionne une police, construite dans TEX, qui ne comporte aucun caractère. TEX l'utilise en remplacement d'une police non définie dans une famille des polices mathématiques.

Styles de caractère

☞ `\bf` utilise un style gras
`\it` utilise un style italique
`\rm` utilise un style romain
`\sl` utilise un style penché
`\tt` utilise un style machine à écrire

Ces commandes choisissent un style de caractère sans changer l'oeil ou la taille du caractère.[2] Normalement vous enfermeriez une de ces commandes de sélection de style de caractère dans un groupe, avec le texte devant être composé dans la police choisie. En dehors d'un groupe une commande de sélection de style est efficace jusqu'à la fin du document (à moins que vous l'effaciez avec une autre commande de ce genre).

Exemple :

```
The Dormouse was {\it not} amused.
```

produit :

The Dormouse was *not* amused.

Voir aussi : "Polices dans des formules mathématiques" (p. 217).

Majuscule et minuscule

`\lccode` ⟨*charcode*⟩ [⟨*nombre*⟩ élément de table]
`\uccode` ⟨*charcode*⟩ [⟨*nombre*⟩ élément de table]

Les valeurs `\lccode` et `\uccode` pour les 256 caractères possibles indiquent la correspondance entre les formes minuscules et majuscules des lettres. Ces valeurs sont utilisées par les commandes `\lowercase` et `\uppercase` respectivement et par l'algorithme de césure de TEX.

TEX initialise les valeurs de `\lccode` et `\uccode` comme suit :

- Le `\lccode` d'une lettre minuscule est le code ASCII de cette lettre.
- Le `\lccode` d'une lettre majuscule est le code ASCII de la lettre minuscule correspondante.
- Le `\uccode` d'une lettre majuscule est le code ASCII de cette lettre.
- Le `\uccode` d'une lettre minuscule est le code ASCII de la lettre majuscule correspondante.

[2] TEX ne fournit pas de commandes prédéfinies pour ne changer que la taille, par exemple, `\eightpoint`. La fourniture de telles commandes exigerait un grand nombre de polices, lesquelles ne seraient jamais utilisées. De telles commandes ont été cependant employées dans la composition de *The TEXbook*.

- Les `\lccode` et `\uccode` d'un caractère qui n'est pas une lettre sont à zéro.

La majeure partie du temps, il n'y a aucune raison de changer ces valeurs, mais vous pourriez devoir les changer si vous vous servez d'une langue qui a plus de lettres que l'anglais.

Exemple :

```
\char\uccode‘s \char\lccode‘a \char\lccode‘M
```

produit :

Sam

`\lowercase {` ⟨*liste de token*⟩ `}`
`\uppercase {` ⟨*liste de token*⟩ `}`

Ces commandes convertissent les lettres de la ⟨*token list*⟩, c'est-à-dire, les tokens avec le code de catégorie 11, dans leurs formes minuscules et majuscules. La conversion d'une lettre est définie par sa valeur de table `\lccode` (pour une minuscule) ou `\uccode` (pour une majuscule). Les tokens de la liste qui ne sont pas des lettres ne sont pas affecter—même si les tokens sont des appels de macro ou d'autres commandes qui se développent en lettres.

Exemple :

```
\def\x{Cd} \lowercase{Ab\x} \uppercase{Ab\x}
```

produit :

abCd ABCd

Espace inter-mot

☞ `\␣`

Cette commande produit explicitement un espace entre les mots appelé "espace contrôlé". un espace contrôlé est utile quand une lettre apparaît juste après une commande, ou dans n'importe quelle autre circonstance où vous ne voulez pas que deux tokens soient reliés dans la sortie. La place produite par `\␣` est indépendante de la ponctuation précédente, c'est-à-dire, que son facteur d'espace (p. 113) est 1000.

Par ailleurs, si vous voulez imprimer le caractère '␣' que nous avons utilisé ici pour noter un espace, vous pouvez l'obtenir en saisissant `{\tt \char ‘\ }`.

Exemple :

```
The Dormouse was a \TeX\ expert, but he never let on.
```

produit :

The Dormouse was a TEX expert, but he never let on.

`\space`
Cette commande est équivalente à la saisie d'un caractère espace. Elle diffère de `\␣` du fait que sa largeur *peut* être affecté par la ponctuation précédente.

Exemple :
```
Yes.\space No.\space Maybe.\par
Yes.\␣No.\␣Maybe.
```

produit :
Yes. No. Maybe.
Yes. No. Maybe.

`^^M`
Cette construction produit le caractère fin de ligne. Elle a normalement deux effets quand TEX la rencontre dans votre source :

1) Elle agit en tant que commande, en introduisant un espace (si elle apparaît à l'extrémité d'une ligne non vide) ou un token `\par` (si elle apparaît à l'extrémité d'une ligne vide).
2) Elle termine la ligne du source, faisant que TEX ignore les caractères restants sur la ligne.

Cependant, `^^M` ne fait *pas* terminer la ligne quand il apparaît dans le contexte `‘\^^M`, dénotant le code ASCII du control-M (le nombre 13). Vous pouvez changer la signification du `^^M` en lui donnant un code de catégorie différent. Voir page 54 pour une explication plus générale de la notation `^^`.

Exemple :
```
Hello.^^MGoodbye.
Goodbye again.\par
The \char ‘\^^M\ character.\par
% The fl ligature is at position 13 of font cmr10
\number ‘\^^M\ is the end of line code.\par
Again, \number ‘^^M is the end of line code,
isn't it? % 32 is the ASCII code for a space
```
produit :
Hello. Goodbye again.
The fl character.
13 is the end of line code.
Again, 32isn't it?

☞ `~`
Le caractère actif ‘~’ appelé un “tilde”, produit une espace normale entre deux mots et lie ces mots pour qu'une coupure de ligne ne se produise pas entre eux. Vous devez employer un tilde dans n'importe quel contexte où une coupure de ligne serait embrouillante, par exemple, avant une

initiale moyenne, après une abréviation telle que "Dr" ou après "Fig." dans "Fig. 8".

Exemple :

```
P.D.Q.~Bach (1807--1742), the youngest and most
imitative son of Johann~S. Bach, composed the
{\sl Concerto for Horn and Hardart}.
```

produit :

P.D.Q. Bach (1807–1742), the youngest and most imitative son of Johann S. Bach, composed the *Concerto for Horn and Hardart*.

☞ `\/`

Chaque caractère d'une fonte TEX a une correction d'"italique" liée à lui, bien que la correction d'italique soit normalement à zéro pour un caractère d'une police non penchée (montante). La correction d'italique indique l'espace supplémentaire qui est nécessaire quand vous passez d'une police inclinée (pas nécessairement une police italique) à une police non penchée. L'espace supplémentaire est nécessaire parce qu'un caractère incliné dépasse dans l'espace qui le suit, réduisant l'espace quand le caractère suivant est non penché. Le dossier de métriques d'une police inclut la correction d'italique de chaque caractère dans la police.

La commande `\/` produit une correction d'italique pour le caractère précédent. Vous devez insérer une correction d'italique quand vous passez d'une police inclinée vers une police droite, sauf quand le caractère suivant est un point ou une virgule.

Exemple :

```
However, {\it somebody} ate {\it something}: that's clear.

However, {\it somebody\/} ate {\it something\/}:
that's clear.
```

produit :

However, *somebody* ate *something*: that's clear.
However, *somebody* ate *something*: that's clear.

`\frenchspacing`
`\nonfrenchspacing`

TEX ajuste normalement l'espacement entre les mots en accord avec les signes de ponctuation. Par exemple, il insère de l'espace supplémentaire à la fin d'une phrase et ajoute un certain étirement au ressort après n'importe quel signe de ponctuation. La commande `\frenchspacing` indique à TEX de rendre l'espacement entre les mots indépendant de la ponctuation, alors que la commande `\nonfrenchspacing` indique à TEX d'utiliser les règles normales d'espacement. Si vous n'indiquez pas `\frenchspacing`, vous obtiendrez l'espacement normal de TEX.

Voyez page 13 pour des conseils sur la façon de contrôler le traitement de la ponctuation de TEX à la fin des phrases.

Exemple :

```
{\frenchspacing  An example: two sentences. Right? No.\par}
{An example: two sentences. Right? No. \par}%
```

produit :

An example: two sentences. Right? No.
An example: two sentences. Right? No.

`\obeyspaces`

TEX condense normalement une suite de plusieurs espaces en un espace simple. `\obeyspaces` demande à TEX de produire un espace dans le résultat pour chaque espace dans le source. Cependant,`\obeyspaces` ne met pas d'espaces au début d'une ligne ; pour cela nous vous recommandons la commande de `\obeywhitespace` définie dans `eplain.tex` (p. 303). `\obeyspaces` est souvent utile quand vous composez quelque chose, un source informatique par exemple, dans une police à espacement fixe (une dans laquelle chaque caractère a le même espacement) et que vous voulez montrer exactement à quoi ressemble chaque ligne du source.

Vous pouvez utiliser la commande `\obeylines` (p. 128) pour obtenir que TEX suive les bords de ligne de votre source. `\obeylines` est souvent utilisé en combinaison avec `\obeyspaces`.

Exemple :

```
These     spaces    are    closed   up
{\obeyspaces but   these  are     not   }.
```

produit :

These spaces are closed up but   these are   not   .

`\spacefactor` [⟨*nombre*⟩ paramètre]
`\spaceskip` [⟨*ressort*⟩ paramètre]
`\xspaceskip` [⟨*ressort*⟩ paramètre]
`\sfcode` ⟨*charcode*⟩ [⟨*nombre*⟩ élément de table]

Ces paramètres primitifs affectent combien d'espace TEX met entre deux mots adjacents, c'est-à-dire, l'espacement inter-mots. L'espacement normal entre les mots est assuré par la police courante. Pendant que TEX traite une liste horizontal, il garde trace du facteur d'espacement f dans `\spacefactor`. pendant qu'il traite chaque caractère du source c, il met à jour f selon la valeur de f_c, le code de facteur d'espacement de c (voir ci- dessous). Pour la plupart des caractères, f_c est à 1000 et TEX met f à 1000. (la valeur initiale de f est également 1000.) Quand TEX voit un espace entre les mots, il ajuste la taille de cet espace en multipliant l'étirement et le rétrécissement de cet espace par $f/1000$ et $1000/f$ respectivement. d'où :

1) Si $f = 1000$, l'espace inter-mots garde sa valeur normale.

2) If $f < 1000$, l'espace inter-mots prend moins d'étirement et plus de rétrécissement.
3) If $f > 1000$, l'espace inter-mots prend plus d'étirement et moins de rétrécissement.

En outre, si $f \geq 2000$ l'espace inter-mots est encore augmenté par le paramètre "d'espace supplémentaire" lié à la police courante.

Chaque caractère saisi c a une adresse dans la table de `\sfcode` (code de facteur d'espacement). L'adresse de la table `\sfcode` est indépendante de la police. Normalement TEX met simplement f à f_c après avoir traité c. De toute manière :

- Si f_c est à zéro, TEX laisse f sans changement. Ainsi, un caractère tel que '`)`' dans plain TEX, pour lequel f_c est à zéro, est essentiellement transparent pour le calcul de l'espacement entre les mots.
- Si $f < 1000 < f_c$, TEX met f à 1000 plutôt que f_c, c'est-à- dire, qu'il refuse d'augmenter f très rapidement.

La valeur de `\sfcode` pour un point est normalement de 3000, c'est pourquoi TEX met habituellement un espace supplémentaire après un point (voir la règle ci-dessus pour le cas $f \geq 2000$). Les caractères non lettre d'une liste horizontale, par exemple, les traits verticaux, agissent généralement comme des caractères avec un facteur d'espace de 1000.

Vous pouvez changer le facteur d'espace explicitement en donnant une valeur numérique différente à `\spacefactor`. Vous pouvez également augmenter l'espacement normal entre les mots en donnant une valeur numérique différente à `\xspaceskip` ou à `\spaceskip` :

- `\xspaceskip` spécifie le ressort utilisé quand $f \geq 2000$; dans le cas où `\xspaceskip` est à zéro, les règles normales s'appliquent.
- `\spaceskip` spécifie le ressort utilisé quand $f < 2000$ ou bien quand `\xspaceskip` est à zéro ; si `\spaceskip` est à zéro, les règles normales s'appliquent. L'étirement et le rétrécissement du ressort `\spaceskip`, comme le ressort ordinaire entre les mots, est modifié selon la valeur de f.

Voir page 76 de *The TEXbook* et 99 de la traduction française pour les règles précises que TEX utilise pour calculer le ressort entre les mots et pages 285–287 de *The TEXbook* et 999-999 de la traduction française pour les ajustements fait à `\spacefactor` après différents items dans une liste horizontale.

Centrer et justifier les lignes

☞ `\centerline` ⟨*argument*⟩
`\leftline` ⟨*argument*⟩
`\rightline` ⟨*argument*⟩

La commande `\centerline` produit un hbox aussi au large que la ligne courante et place ⟨*argument*⟩ au centre de la boîte. Les commandes `\leftline` et `\rightline` sont analogues ; elles placent ⟨*argument*⟩ au bord gauche ou droite de la boîte. Si vous voulez appliquer une de ces commandes à plusieurs lignes consécutives, vous devez l'appliquer à chacune individuellement. Voir page 316 pour une autre approche.

N'utilisez pas ces commandes dans un paragraphe—si vous le faites, TEX ne pourra probablement pas couper le paragraphe en lignes et se plaindra au sujet d'un overfull hbox.

Exemple :

```
\centerline{Grand Central Station}
\leftline{left of Karl Marx}
\rightline{right of Genghis Khan}
```

produit :

Grand Central Station

left of Karl Marx

right of Genghis Khan

☞ `\line` ⟨*argument*⟩

Cette commande produit un hbox contenant l'⟨*argument*⟩. Cette hbox est exactement aussi large que la ligne courante, c'est-à-dire, qu'il s'étend de la marge droite à la gauche.

Exemple :

```
\line{ugly \hfil suburban \hfil sprawl}
% Without \hfil you'd get an 'underfull box' from this.
```

produit :

ugly suburban sprawl

`\llap` ⟨*argument*⟩
`\rlap` ⟨*argument*⟩

Ces commandes vous permettent de produire un texte qui recouvre celui qui est à gauche ou à droite de la position actuelle. `\llap` recule par la largeur de l'⟨*argument*⟩ et puis compose l'⟨*argument*⟩. le `\rlap` est semblable, sauf qu'il compose l'⟨*argument*⟩ d'abord et recule. `\llap` et `\rlap` sont utiles pour placer du texte en dehors des marges courantes. `\llap` et `\rlap` effectuent leur travail en créant un box de largeur zéro.

Vous pouvez également utiliser `\llap` ou `\rlap` pour construire des caractères spéciaux par surimpression, mais ne l'essayez pas à moins d'être sûr que les caractères que vous utilisez aient la même largeur (ce qui est le cas pour une police à espacement fixe telle que `cmtt10`, la Computer Modern 10-points police de type machine à écrire)..

Exemple :

```
\noindent\llap{off left }\line{\vrule $\Leftarrow$
left margin of examples\hfil right margin of examples
$\Rightarrow$\vrule}\rlap{ off right}
```

produit :

off left |⇐ left margin of examples right margin of examples ⇒| off right

Voir aussi : `\hsize` (p. 120).

Formation des paragraphes

débuter, finir et indenter des paragraphes

`\par`

Cette commande termine un paragraphe et met TEX dans le mode vertical, prêt à ajouter plus d'articles sur la page. Puisque TEX convertis une ligne blanche dans votre fichier source en un token `\par`, vous n'avez normalement pas besoin de saisir `\par` explicitement pour finir un paragraphe.

Un point important est que `\par` n'indique pas à TEX de commencer un paragraphe ; il ne lui dit que de le finir. TEX débute un paragraphe quand il est en mode vertical ordinaire (tel qu'il est après un `\par`) et rencontre un article en soi horizontal tel qu'une lettre. En tant qu'élément du cérémonial de début de paragraphe, TEX insère une quantité d'espace vertical donnée par le paramètre `\parskip` (p. 147) et indente le paragraphe par un espace horizontal donné par `\parindent` (p. 119).

Vous pouvez normalement effacer tout l'espace inter-paragraphe produit avec `\par` en mettant la commande `\vskip -\lastskip`. Caci peut être utile quand vous écrivez une macro censé travailler de la même manière si elle est précédée ou non par une ligne blanche.

Vous pouvez obtenir que TEX prenne certaines mesures spéciales au début de chaque paragraphe en plaçant des instructions dans `\everypar` (p. 119).

Voir pages 283 et 286 de *The TEXbook* et 999 et 999 de la traduction française pour les effets précis de `\par`.

Exemple :

```
\parindent = 2em
``Can you row?'' the Sheep asked, handing Alice a pair of
knitting-needles as she was speaking.\par ``Yes, a little%
---but not on land---and not with needles---'' Alice was
starting to say, when suddenly the needles turned into oars.
```

produit :

"Can you row?" the Sheep asked, handing Alice a pair of knitting-needles as she was speaking.

"Yes, a little—but not on land—and not with needles—" Alice was starting to say, when suddenly the needles turned into oars.

`\endgraf`

Cette commande est un synonyme de la commande primitive `\par`. Elle est utile quand vous avez redéfini `\par` mais voulez toujours avoir accès à la définition originale de `\par`.

`\parfillskip` [⟨*ressort*⟩ paramètre]

Ce paramètre désigne le ressort horizontal que TEX insert à la fin d'un paragraphe. La valeur par défaut de `\parfillskip` est `0pt plus 1fil`, qui fait que la dernière ligne d'un paragraphe soit complétée de l'espace vide. Une valeur de `0pt` force TEX à finir la dernière ligne d'un paragraphe sur la marge droite.

☞ `\indent`

Si TEX est en mode verticale, comme il l'est après avoir fini un paragraphe, cette commande insère le ressort inter-paragraphe `\parskip`, met TEX en mode horizontal, débute un paragraphe et indente ce paragraphe de `\parindent`. Si TEX est déjà en mode horizontal, cette commande produit simplement un espace blanc de largeur `\parindent`. Deux `\indent` à la suite produisent deux indentations.

Comme dans l'exemple ci-dessous, un `\indent` à un endroit où TEX commencerait de toute façon un paragraphe est superflu. Quand TEX est en mode vertical et voit une lettre ou une autre commande en soi horizontale, il commence un paragraphe en commutant vers le mode horizontal, fait un `\indent` et traite la commande horizontale.

Exemple :

```
\parindent = 2em  This is the first in a series of three
paragraphs that show how you can control indentation. Note
that it has the same indentation as the next paragraph.\par
\indent This is the second in a series of three paragraphs.
It has \indent an embedded indentation.\par
\indent\indent This doubly indented paragraph
is the third in the series.
```

produit :

This is the first in a series of three paragraphs that show how you can control indentation. Note that it has the same indentation as the next paragraph.

This is the second in a series of three paragraphs. It has an embedded indentation.

This doubly indented paragraph is the third in the series.

☞ `\noindent`

Si le TEX est en mode vertical, comme il l'est après la fin d'un paragraphe, cette commande insère le ressort inter-paragraphe `\parskip`, met TEX dans le mode horizontal, et commence un paragraphe non indenté. Elle n'a aucun effet dans le mode horizontal, c'est-à-dire, dans un paragraphe. Commencer un paragraphe avec `\noindent` décommande ainsi l'indentation de `\parindent` qui se produirait normalement.

Une utilisation commune de `\noindent` est d'interdire l'indentation de la première ligne d'un paragraphe quand le paragraphe suit un certain matériel affiché.

Exemple :

```
\parindent = 1em
Tied round the neck of the bottle was a label with the
words \smallskip \centerline{EAT ME}\smallskip
\noindent beautifully printed on it in large letters.
```

produit :

Tied round the neck of the bottle was a label with the words

EAT ME

beautifully printed on it in large letters.

`\textindent` ⟨*argument*⟩

Cette commande indique à TEX de commencer un paragraphe et de l'indenter de `\parindent`, comme d'habitude. TEX alors justifie à droite l'⟨*argument*⟩ de l'indentation et le fait suivre d'un espace demi-cadratin (moitié d'un cadratin). Plain TEX utilise cette commande pour composer les notes de pied de page. (p. 151) et les articles dans les listes (voir `\item`, p. 136).

Exemple :

```
\parindent = 20pt \textindent{\raise 1pt\hbox{$\bullet$}}%
You are allowed to use bullets in \TeX\ even if
you don't join the militia, and many peace-loving
typographers do so.
```

produit :

• You are allowed to use bullets in TEX even if you don't join the militia, and many peace-loving typographers do so.

`\parindent` [⟨*dimension*⟩ paramètre]

Ce paramètre indique la quantité par laquelle la première ligne de chaque paragraphe doit être indentée. Comme montré dans l'exemple ci-dessous, placer `\parindent` et `\parskip` à zéro est une mauvaise idée puisque alors les coupures de paragraphe ne sont plus évidente.

Exemple :

```
\parindent = 2em This paragraph is indented by 2 ems.
\par \parindent=0pt This paragraph is not indented at all.
\par Since we haven't reset the paragraph indentation,
this paragraph isn't indented either.
```

produit :

This paragraph is indented by 2 ems.
This paragraph is not indented at all.
Since we haven't reset the paragraph indentation, this paragraph isn't indented either.

`\everypar` [⟨*liste de token*⟩ paramètre]

TEX exécute les commandes des ⟨*token list*⟩ toutes les fois qu'il entre dans le mode horizontal, par exemple, quand il commence un paragraphe. Par défaut `\everypar` est vide, mais vous pouvez prendre des mesures supplémentaires au début de chaque paragraphe en mettant des commandes pour ces actions dans un token list. et assigner ce token list à `\everypar`.

Exemple :

```
\everypar = {$\Longrightarrow$\enspace}
Now pay attention!\par
I said, ``Pay attention!''.\par
I'll say it again! Pay attention!
```

produit :

⟹ Now pay attention!
⟹ I said, "Pay attention!".
⟹ I'll say it again! Pay attention!

Formation de paragraphes entiers

☞ `\hsize` [⟨*dimension*⟩ paramètre]

Ce paramètre indique la longueur courante de la ligne, c'est-à- dire, la largeur habituelle des lignes dans un paragraphe commençant à la marge gauche. De grands beaucoup de TEX commandes, par exemple, `\centerline` (p. 115) et `\hrule` (p. 178), emploient implicitement la valeur du `\hsize`. En changeant le `\hsize` chez un groupe vous pouvez changer la largeur des constructions produites par de telles commandes.

Si vous placez `\hsize` dans un vbox qui contient du texte, le vbox aura la largeur que vous aurez donnée au `\hsize`.

Plain TEX fixe `\hsize` à `6.5in`.

Exemple :

```
{\hsize = 3.5in % Set this paragraph 3.5 inches wide.
The hedgehog was engaged in a fight with another hedgehog,
which seemed to Alice an excellent opportunity for
croqueting one of them with the other.\par}%
```

produit :

The hedgehog was engaged in a fight with another hedgehog, which seemed to Alice an excellent opportunity for croqueting one of them with the other.

Exemple :

```
\leftline{\raggedright\vtop{\hsize = 1.5in
Here is some text that we put into a paragraph that is
an inch and a half wide.}\qquad
\vtop{\hsize = 1.5in Here is some more text that
we put into another paragraph that is an inch and a
half wide.}}
```

produit :

Here is some text that we put into a paragraph that is an inch and a half wide.

Here is some more text that we put into another paragraph that is an inch and a half wide.

☞ `\narrower`

Cette commande donne des paragraphes plus étroits, en augmentant les marges gauches et droites de `\parindent`, l'indentation de paragraphe courante. Elle réalise ceci en augmentant `\leftskip` et `\rightskip` de `\parindent`. Normalement vous placez `\narrower` au début d'un groupe contenant les paragraphes que vous voulez rendre plus étroits. Si vous oubliez d'enfermer `\narrower` dans un groupe, vous constaterez que tout le reste de votre document aura des paragraphes étroits.

`\narrower` n'affecte que les paragraphes qui terminent après que vous l'ayez appelé. Si vous terminez un groupe `\narrower` avant d'avoir fini un paragraphe, TEX ne rendra pas ce paragraphe plus étroit.

Exemple :

```
{\parindent = 12pt \narrower\narrower\narrower
This is a short paragraph. Its margins are indented
three times as much as they would be
had we used just one ``narrower'' command.\par}
```

produit :

> This is a short paragraph. Its margins are indented three times as much as they would be had we used just one "narrower" command.

`\leftskip` [⟨*ressort*⟩ paramètre]
`\rightskip` [⟨*ressort*⟩ paramètre]

Ces paramètres indiquent à TEX combien de ressort placer aux extrémités gauches et à droites de chaque ligne du paragraphe courant. Nous n'expliquerons que le fonctionnement de `\leftskip` puisque `\rightskip` est similaire.

Vous pouvez augmenter la marge gauche en plaçant `\leftskip` à une dimension différente de zéro. Si vous donnez à `\leftskip` un certain étirement, vous pouvez produire du texte cadré à droite, c'est-à-dire, du texte qui a une marge gauche inégale.

Normalement, vous devriez enfermer tout assignement à `\leftskip` dans un groupe avec le texte affecté afin d'empêcher son effet de continuer jusqu'à la fin de votre document. Cependant, il est injustifié de changer la valeur de `\leftskip` à l'intérieur d'un groupe qui est, lui aussi, contenu dans un paragraphe—la valeur de `\leftskip` à la *fin* d'un paragraphe est celle qui détermine comment TEX coupera le paragraphe en lignes.

Exemple :

```
{\leftskip = 1in The White Rabbit trotted slowly back
again, looking anxiously about as it went, as if it had
lost something.  {\leftskip = 10in % has no effect
It muttered to itself, ``The Duchess! The Duchess! She'll
get me executed as sure as ferrets are ferrets!''}\par}%
```

produit :

> The White Rabbit trotted slowly back again, looking anxiously about as it went, as if it had lost something. It muttered to itself, "The Duchess! The Duchess! She'll get me executed as sure as ferrets are ferrets!"

Exemple :

```
\pretolerance = 10000 % Don't hyphenate.
\rightskip = .5in plus 2em
The White Rabbit trotted slowly back again, looking
anxiously about as it went, as if it had lost something.
It muttered to itself, ``The Duchess! The Duchess! She'll
get me executed as sure as ferrets are ferrets!''
```

produit :

> The White Rabbit trotted slowly back again, looking anxiously about as it went, as if it had lost something. It muttered to itself, "The Duchess! The Duchess! She'll get me executed as sure as ferrets are ferrets!"

☞ `\raggedright`
`\ttraggedright`

Ces commandes font que TEX compose votre document "cadré à gauche". Les espaces inter-mots ont tous leur taille naturelle, c'est-à-dire, elles ont toutes la même largeur et ne s'étirent ni se rétrécissent. En conséquence, leur marge droite n'est généralement pas la même. L'alternative, qui est le comportement par défaut de TEX, est de composer votre document justifié, c'est-à-dire, avec des marges gauches et droites uniformes. Dans du texte justifié, les espaces inter-mots sont étirés pour aligner des marges de droite. Certains typographes préfèrent cadrer à gauche parce que cela évite des "rivières" distractive d'espace sur la page imprimée.

Vous devez utiliser la commande `\ttraggedright` pour composer du texte dans une police non proportionnelle et la commande `\raggedright` command pour composer de texte dans toutes les autres polices.

La plupart du temps vous voudrez appliquer ces commandes à un document entier, mais vous pouvez limiter leurs effets en les englobant dans un groupe.

Exemple :

```
\raggedright ``You couldn't have it if you {\it did\/}
want it,'' the Queen said. ``The rule is, jam tomorrow
and jam yesterday---but never jam {\it today\/}.''
``It {\it must\/} come sometimes to `jam today,%
thinspace'' Alice objected. ``No, it can't'', said the
Queen. ``It's jam every {\it other\/} day: today isn't
any {\it other\/} day.''
```

produit :

"You couldn't have it if you *did* want it," the Queen said. "The rule is, jam tomorrow and jam yesterday—but never jam *today*." "It *must* come sometimes to 'jam today,'" Alice objected. "No, it can't", said the Queen. "It's jam every *other* day: today isn't any *other* day."

`\hang`

Cette commande indente la seconde ligne et les suivantes d'un paragraphe de `\parindent`, l'indentation du paragraphe. (p. 119). Puisque la première ligne est déjà indentée par `\parindent` (à moins que vous ayez supprimé l'indentation avec `\noindent`), le paragraphe entier apparaît être indenté de `\parindent`.

Exemple :

```
\parindent=24pt \hang  ``I said you {\it looked} like an
egg, Sir,'' Alice gently explained to Humpty Dumpty. ``And
some eggs are very pretty, you know,'' she added.
```

produit :

"I said you *looked* like an egg, Sir," Alice gently explained to Humpty Dumpty. "And some eggs are very pretty, you know," she added.

`\hangafter` [⟨*nombre*⟩ paramètre]
`\hangindent` [⟨*dimension*⟩ paramètre]

Ces deux paramètres associés spécifient l'"indentation rémanente" pour un paragraphe. L'indentation rémanente indique à TEX que certaines lignes du paragraphe doivent être indentées et que les lignes restantes doivent avoir leur largeur normale `\hangafter` détermine quelles lignes sont indentées, tandis que `\hangindent` détermine le montant d'indentation et si elle apparaît sur la gauche ou sur la droite :

- Soit n la valeur de `\hangafter`. Si $n < 0$, Les $-n$ premières lignes du paragraphe seront indentées. Si $n \geq 0$, toutes sauf les n premières lignes du paragraphe seront indentées.
- Soit x la valeur de `\hangindent`. Si $x \geq 0$, les lignes seront indentées de x à gauche. Si $x < 0$, les lignes seront indentées de $-x$ sur la droite.

Quand vous spécifiez de l'indentation rémanente, elle ne s'applique qu'au paragraphe suivant (si vous êtes en mode vertical) ou au paragraphe courant (si vous êtes en mode horizontal). TEX utilise les valeurs de `\hangafter` et `\hangindent` à la fin d'un paragraphe, quand il coupe ce paragraphe en lignes.

à la différence de la plupart des autres paramètres de mise en forme de paragraphe, `\hangafter` et `\hangindent` sont réinitialisés au début de chaque paragraphe, soit, 1 pour `\hangafter` et 0 pour `\hangindent`. Si vous voulez composer une suite de paragraphes avec de l'indentation rémanente, utilisez `\everypar` (p. 119). Si vous spécifiez `\hangafter` et `\hangindent` de même que `\parshape`, TEX ignorera `\hangafter` et `\hangindent`.

Exemple :

```
\hangindent=6pc \hangafter=-2
This is an example of a paragraph with hanging indentation.
In this case, the first two lines are indented on the left,
but after that we return to unindented text.
```

produit :

This is an example of a paragraph with hanging indentation. In this case, the first two lines are indented on the left, but after that we return to unindented text.

Exemple :

```
\hangindent=-6pc \hangafter=1
This is another example of a paragraph with hanging
indentation.  Here, all lines after the first have been
indented on the right. The first line, on the other
hand, has been left unindented.
```

produit :

This is another example of a paragraph with hanging indentation. Here, all lines after the first have been indented on the right.
The first line, on the other hand, has been left unindented.

`\parshape` $n\ i_1 l_1\ i_2 l_2\ \ldots\ i_n l_n$

Cette commande spécifie le gabarit des n premières lignes d'un paragraphe— Le prochain si vous êtes en mode vertical et le paragraphe courant si vous êtes en mode horizontal. Les i et les l sont tous des dimensions. La première ligne est indentée de i_1 et a une longueur de l_1, la seconde ligne est indentée de i_2 et a une longueur de l_2, et ainsi de suite. si le paragraphe a plus de n lignes, la dernière paire indentation/longueur est utilisée pour les autres lignes. Pour parachever des effets spéciaux comme celui montré ici, vous devrez normalement faire beaucoup d'essai, insérer des crénages ici et là et choisir votre mot pour remplir le gabarit.

\parshape, comme \hangafter et \hangindent, n'est effectif que pour un paragraphe. Si vous spécifiez \hangafter et \hangindent de même que \parshape, TEX ignorera \hangafter et \hangindent.

Exemple :

```
% A small font and close interline spacing make this work
\smallskip\font\sixrm=cmr6 \sixrm \baselineskip=7pt
\fontdimen3\font = 1.8pt \fontdimen4\font = 0.9pt
\noindent \hfuzz 0.1pt
\parshape 30 0pt 120pt 1pt 118pt 2pt 116pt 4pt 112pt 6pt
108pt 9pt 102pt 12pt 96pt 15pt 90pt 19pt 84pt 23pt 77pt
27pt 68pt 30.5pt 60pt 35pt 52pt 39pt 45pt 43pt 36pt 48pt
27pt 51.5pt 21pt 53pt 16.75pt 53pt 16.75pt 53pt 16.75pt 53pt
16.75pt 53pt 16.75pt 53pt 16.75pt 53pt 16.75pt 53pt 16.75pt
53pt 14.6pt 48pt 24pt 45pt 30.67pt 36.5pt 51pt 23pt 76.3pt
Les vins de France et de Californie semblent \^etre
les plus connus, mais ne sont pas les seuls bons vins.
Les vins espagnols sont souvent sous-estim\'es,
et des assez vieux peu\-vent \^etre disponibles \`a des
prix raisonnables. Pour les vins espagnols, le mill\'esime
n'est pas important, mais le climat de la r\'egion de
Bordeaux varie selon les ann\'ees. Seuls certains sont
bons. Ceux que
vo\kern -.1pt us de\kern -.1pt v\kern -.1pt ez
noter tr\`es tr\`es bons~:
1962, 1964, 1966. 1958, 1959, 1960, 1961, 1964, 1966 sont de
bons crus
californien. \`A boire avec mod\'eration~
```

produit :

Les vins de France et de Californie semblent être les plus connus, mais ne sont pas les seuls bons vins. Les vins espagnols sont souvent sous-estimés, et des assez vieux peuvent être disponibles à des prix raisonnables. Pour les vins espagnols, le millésime n'est pas important, mais le climat de la région de Bordeaux varie selon les années. Seuls certains sont bons. Ceux que vous devez noter très très bons : 1962, 1964, 1966. 1958, 1959, 1960, 1961, 1964, 1966 sont de bons crus californien. À boire avec modération !

`\prevgraf` [⟨*nombre*⟩ paramètre]
En mode horizontal, ce paramètre spécifie le nombre de lignes dans le paragraphe en cours ; en mode vertical, il spécifie le nombre de lignes dans le paragraphe précédent. TEX ne prend en compte `\prevgraf` qu'après avoir fini de couper du texte en ligne, c'est-à-dire, sur un affichage mathématique ou à la fin d'un paragraphe. Voir la page 103 de *The TEXbook* et 999 de la traduction française pour plus de détails à son propos.

`\vadjust {` ⟨*matériel en mode vertical*⟩ `}`
Cette commande insère le ⟨*matériel en mode vertical*⟩ spécifié juste après la ligne de sortie contenant la position où la commande se trouve. Vous pouvez l'utiliser, par exemple, pour provoquer un saut de page ou insérer de l'espace supplémentaire après une certaine ligne.

Exemple :

```
Some of these words are \vadjust{\kern8pt\hrule} to be
found above the line and others are to be found below it.
```

produit :

Some of these words are to be found above the line and others are to

be found below it.

Voir aussi : `\parindent` (p. 119), `\parskip` (p. 147) ainsi que `\everypar` (p. 119).

coupures de lignes

Encourager ou décourager les coupures de ligne

`\break`
Cette commande force une coupure de ligne. à moins que vous fassiez quelque chose pour remplir la ligne, Vous recevrez vraisemblablement une plainte "underfull hbox". `\break` peut aussi être utilisé en mode vertical.

Exemple :

```
Fill out this line\hfil\break and start another one.\par
% Use \hfil here to fill out the line.
This line is underfull---we ended it\break prematurely.
% This line causes an 'underfull hbox' complaint.
```

produit :

Fill out this line
and start another one.
This line is underfull—we ended it prematurely.

`\nobreak`

Cette commande empêche une coupure de ligne là où elle devrait se faire normalement. `\nobreak` peut aussi être utilisée en mode vertical.

Exemple :

```
Sometimes you'll encounter a situation where
a certain space\nobreak\qquad must not get lost.
```

produit :

Sometimes you'll encounter a situation where a certain space must not get lost.

`\allowbreak`

Cette commande demande à TEX d'autoriser une coupure de ligne là où elle n'arriverait pas normalement. Elle est plus souvent utile dans une formule mathématique, car TEX est réticent à couper de telles lignes. `\allowbreak` peut aussi être utilisée en mode vertical.

Exemple :

```
Under most circumstances we can state with some confidence
that $2+2\allowbreak=4$, but skeptics may disagree.
\par For such moronic automata, it is not difficult to
analyze the input/\allowbreak output behavior in the limit.
```

produit :

Under most circumstances we can state with some confidence that $2+2 = 4$, but skeptics may disagree.

For such moronic automata, it is not difficult to analyze the input/ output behavior in the limit.

`\penalty` ⟨*nombre*⟩

Cette commande produit un élément de pénalité. L'élément de pénalité rend TEX plus ou moins désireux de couper une ligne à l'endroit où cet élément est placé. Une pénalité négative, c'est-à-dire, un bonus, encourage une coupure de ligne ; une pénalité positive décourage une coupure de ligne. Une pénalité de 10000 ou plus empêche une coupure quel qu'elle soit, tandis qu'une pénalité de -10000 ou moins force une coupure. `\penalty` peut aussi être utilisée en mode vertical.

Exemple :

```
\def\break{\penalty -10000 } % as in plain TeX
\def\nobreak{\penalty 10000 } % as in plain TeX
\def\allowbreak{\penalty 0 } % as in plain TeX
```

`\obeylines`
TEX normalement traite une fin de ligne comme un espace. `\obeylines` demande à TEX de traiter chaque fin de ligne comme une fin de paragraphe, forçant ainsi une coupure de ligne. `\obeylines` est souvent utile quand vous composez un poème ou un programme informatique. Si quelques lignes sont plus longues que la longueur de ligne réelle (`\hsize` − `\parindent`), sinon, vous aurez une coupure de ligne supplémentaire dans ces lignes.

Parce que TEX insère le ressort `\parskip` (p. 147) entre des lignes contrôlée par `\obeylines` (puisqu'il pense que chaque ligne est un paragraphe), vous devrez normalement mettre `\parskip` à zéro quand vous utiliserez `\obeylines`.

Vous pouvez utiliser la commande `\obeyspaces` (p. 113) pour que TEX prenne en compte tous les espaces d'une ligne. `\obeylines` et `\obeyspaces` sont souvent utilisées ensemble.

Exemple :
```
\obeylines
``Beware the Jabberwock, my son!
\quad The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
\quad The frumious Bandersnatch!''
```
produit :

"Beware the Jabberwock, my son!
  The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
  The frumious Bandersnatch!"

☞ `\slash`
Cette commande produit un slash (/) et prévient aussi TEX qu'il peut couper la ligne après le slash, si nécessaire.

Exemple :
```
Her oldest cat, while apparently friendly to most people,
had a Jekyll\slash Hyde personality when it came to mice.
```
produit :

Her oldest cat, while apparently friendly to most people, had a Jekyll/Hyde personality when it came to mice.

paramètres de coupure de lignes

`\pretolerance` [⟨*nombre*⟩ paramètre]
`\tolerance` [⟨*nombre*⟩ paramètre]
Ces paramètres déterminent la médiocrité que TEX tolèrera sur chaque ligne quand il choisit des coupures de ligne pour un paragraphe. La médiocrité est une mesure de combien l'espacement inter-mot dévie de

l'idéal. `\pretolerance` spécifie la médiocrité tolérable pour des coupure de ligne sans césure ; `\tolerance` spécifie la médiocrité tolérable pour des coupures de ligne avec césure. La médiocrité tolérable peut être dépassée de deux façons : un ligne est trop serrée (les espaces inter-mots sont trop petits) ou trop lâche (les espaces inter-mots sont trop grands).

- Si TEX doit faire une ligne trop relâchée, il se plaint d'un "underfull hbox".
- Si TEX doit faire une ligne trop resserrée, il laisse la ligne dépasser dans la marge droite et se plaint d'un "overfull hbox".

TEX choisit des coupures de ligne selon les étapes suivantes :

1) Il essaye de choisir des coupures de ligne sans césures. Si aucune des lignes résultantes n'a de médiocrité dépassant `\pretolerance`, les coupures de ligne sont acceptées et le paragraphe peut être fait.
2) Sinon, il essaye un autre jeu de coupures de ligne, cette fois en autorisant les césures. Si aucune des lignes résultantes n'a une médiocrité dépassant `\tolerance`, le nouveau jeu de coupures de ligne est acceptable et le paragraphe peut maintenant être fait.
3) Autrement, il ajoute `\emergencystretch` (voir plus bas) à l'étirement de chaque ligne et essaye encore.
4) Si aucun de ces essais n'a produit de jeu de coupures de ligne acceptable, il fait le paragraphe avec un ou plusieurs "overfull hbox" et s'en plaint.

Plain TEX initialise `\tolerance` à 200 et `\pretolerance` à 100. Si vous mettez `\tolerance` à 10000, TEX devient infiniment tolérant et accepte tout espacement, quelque soit sa laideur (à moins qu'il rencontre un mot qui ne tienne pas sur une ligne, même avec césure). Ainsi en changeant `\tolerance` vous pouvez éviter des "overfull hbox" et "underfull hbox", mais au prix de mauvais espacements. En rendant `\pretolerance` plus grand vous pouvez faire que TEX évite les césure (et s'exécute aussi plus rapidement), mais, encore, au prix d'éventuels mauvais espacements. Si vous mettez `\pretolerance` à -1, TEX n'essayera même pas de faire le paragraphe sans césure.

Le paramètre `\hbadness` (p. 176) détermine le niveau de médiocrité que TEX tolèrera avant de se plaindre, mais `\hbadness` n'affecte pas la manière dont TEX compose votre document. Le paramètre `\hfuzz` (p. 176) détermine le montant dont une hbox peut dépasser sa largeur spécifiée avant que TEX la considère erronée.

`\emergencystretch` [⟨*dimension*⟩ paramètre]

En mettant ce paramètre supérieur à zéro, vous pouvez rendre plus facile à TEX la composition de votre document sans générer d'"overfull hbox". C'est une meilleur alternative à `\tolerance=10000`, car cela tend à produire des lignes réellement laides. Si TEX ne peut pas composer un paragraphe sans dépasser `\tolerance`, il tentera encore, en ajoutant

`\emergencystretch` à l'étirement de chaque ligne. L'effet du changement est de réduire la médiocrité de chaque ligne, autorisant TEX à faire des espaces plus larges qu'ils auraient été autrement et ainsi de choisir des coupures de ligne qui seront aussi bonne que possible selon les circonstances.

`\looseness` [⟨*nombre*⟩ paramètre]

Ce paramètre vous donne un moyen de changer le nombre total de lignes dans un paragraphe par rapport à celui qu'il aurait été de manière optimale. `\looseness` est ainsi nommé parce que c'est une mesure de combien perd le paragraphe, c'est-à-dire, combien d'espace supplémentaire il contient.

Normalement, `\looseness` est à 0 et TEX choisit des coupures de ligne selon sa manière habituelle. Mais si `\looseness` est à, disons, 3, TEX fait ainsi :

1) Il choisit des coupures de ligne normalement, obtenant un paragraphe de n lignes.
2) Il écarte ces coupures de ligne et tente de trouver un nouveau jeu de coupure de ligne qui donne le paragraphe en $n + 3$ lignes. (Sans l'étape précédente, TEX ne saurait pas la valeur de n.)
3) Si l'essai précédent donne des lignes dont la médiocrité dépasse `\tolerance`, il tente d'obtenir $n + 2$ lignes—et si cela échoue aussi, $n + 1$ lignes, et finalement n lignes encore.

De même, si `\looseness` est à $-n$, TEX tente de faire le paragraphe avec n lignes de moins que la normale. Le moyen le plus simple pour TEX de faire un paragraphe plus long d'une ligne est de mettre un seul mot sur la ligne en plus. vous pouvez empêcher cela en mettant un tilde (p. 111) entre les deux derniers mots du paragraphe.

Mettre `\looseness` est le meilleur moyen de forcer un paragraphe à occuper un nombre de lignes donné. Le mettre a une valeur négative est pratique quand vous essayez d'augmenter la quantité de texte que vous pouvez placer que une page. Similairement, le mettre à une valeur positive est pratique quand vous essayez de diminuer la quantité de texte sur une page.

TEX remet `\looseness` à 0 quand il termine un paragraphe, après avoir coupé le paragraphe en lignes. Si vous voulez changer le relâchement de plusieurs paragraphes, vous devez le faire individuellement pour chacun ou mettre le changement dans `\everypar` (p. 119).

`\linepenalty` [⟨*nombre*⟩ paramètre]

Ce paramètre spécifie les démérites que TEX répartit pour chaque coupure de ligne quand il coupe un paragraphe en lignes. La pénalité est indépendante d'où la coupure de ligne à lieu. Augmenter la valeur de ce paramètre demande à TEX d'essayer plus fortement de mettre un paragraphe avec un nombre minimum de lignes, même au coût d'autres considérations

esthétiques comme éviter des espacements inter-mots excessivement resserré. Les démérites sont en unités de médiocrité au carré, donc vous devez assigner une valeur plutôt large à ce paramètre (dans les milliers) pour qu'elle ait un quelconque effet. Plain TEX mets `\linepenalty` à 10.

`\adjdemerits` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie des démérites additionnels que TEX attache au point de coupure entre deux lignes adjacentes qui sont "incompatible visuellement". De telles paires de lignes font qu'un paragraphe apparaît inégal. Les incompatibilités sont évaluées en termes d'étroitesse ou de relâchement de lignes :

1) Une ligne est resserrée si son ressort doit se rétrécir d'au moins 50%.
2) Une ligne est décente si sa médiocrité est de 12 ou moins.
3) Une ligne est relâchée si son ressort doit s'étirer d'au moins 50%.
4) Une ligne est très relâchée si son ressort doit tellement s'étirer que sa médiocrité dépasse 100.

Deux lignes adjacentes sont visuellement incompatible si leurs catégories ne sont pas adjacentes, c'est-à-dire, une ligne resserrée est après une relâchée ou une ligne décente après une très relâchée.

Les démérites sont en unités de minrefmédiocrité au carrée, donc vous devez assigner une valeur plutôt large à ce paramètre (dans les milliers) pour qu'elle ait un quelconque effet. Plain TEX mets `\adjdemerits` à 10000.

`\exhyphenpenalty` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie la pénalité que TEX attache à un point de coupure sur une césure explicite comme celle de "helter-skelter". Augmenter ce paramètre a l'effet de décourager TEX de finir une ligne sur une césure explicite. Plain TEX mets `\exhyphenpenalty` à 50.

`\hyphenpenalty` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie la pénalité que TEX attache à un point de coupure sur une césure implicite. des césures implicites peuvent survenir du dictionnaire de césure de TEX ou de césures optionnelles que vous avez inséré avec `\-` (p. 132). Augmenter ce paramètre a l'effet de décourager TEX de couper des mots. Plain TEX mets `\hyphenpenalty` à 50.

`\doublehyphendemerits` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie des démérites additionnels que TEX attache à un point de coupure quand ce point de coupure arrive à deux lignes consécutives qui se termines sur une césure. Augmenter la valeur de ce paramètre a l'effet de décourager TEX de césurer deux lignes à suivre. Les démérites sont en unités de médiocrité au carré, donc vous devez assigner une valeur plutôt large à ce paramètre (dans les milliers) pour qu'elle ait un quelconque effet. Plain TEX mets `\doublehyphendemerits` à 10000.

`\finalhyphendemerits` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie des démérites additionnels que TEX attache à un point de coupure qui fait que l'avant dernière ligne d'un paragraphe se termine avec une césure. Une telle césure est généralement considérée inesthétique parce que l'espace blanc éventuel provoqué par une dernière ligne courte la rend méprisable. Augmenter la valeur de ce paramètre a le effet de décourager TEX de finir l'avant dernière ligne avec une césure. Les démérites sont en unités de médiocrité au carré, donc vous devez assigner une valeur plutôt large à ce paramètre (dans les milliers) pour qu'elle ait un quelconque effet. Plain TEX mets `\finalhyphendemerits` à 5000.

`\binoppenalty` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie la pénalité d'une coupure d'une formule mathématique après un opérateur binaire quand la formule apparaît dans un paragraphe. Plain TEX mets `\binoppenalty` à 700.

`\relpenalty` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie la pénalité pour une coupure d'une formule mathématique après une relation quand la formule apparaît dans un paragraphe. Plain TEX mets `\relpenalty` à 500.

Césure

☞ `\-`
La commande `\-` insère une "césure optionnelle" dans un mot. La césure optionnelle autorise TEX de couper le mot à cet endroit. TEX n'est pas obligé de couper là—il ne le fera que s'il le faut. Cette commande est utile quand un mot qui apparaît une ou deux fois dans votre document doit être coupé, mais que TEX ne peut pas trouver de point de coupure appropriée de lui-même.

Exemple :

```
Alice was exceedingly reluctant to shake hands first
with either Twee\-dle\-dum or Twee\-dle\-dee, for
fear of hurting the other one's feelings.
```

produit :

Alice was exceedingly reluctant to shake hands first with either Tweedledum or Tweedledee, for fear of hurting the other one's feelings.

`\discretionary {` ⟨*pre-break text*⟩ `} {` ⟨*post-break text*⟩ `} {` ⟨*no-break text*⟩ `}`
Cette commande spécifie une "coupure optionnelle", autrement dit, un endroit où TEX peut couper une ligne. Elle dit aussi à TEX quel texte mettre de chaque coté de la coupure.

- Si TEX ne doit pas couper là, il utilise le ⟨*no-break text*⟩.
- Si TEX doit couper là, il mets le ⟨*pre-break text*⟩ just avant la coupure et le ⟨*post-break text*⟩ juste après la coupure.

Comme avec `\-`, TEX n'est pas obligé de couper une ligne sur une coupure optionnelle. En fait, `\-` est normalement équivalent à `\discretionary {-}{}{}`.

Parfois, TEX insère des coupure optionnelle de lui-même. Par exemple, il insère `\discretionary{}{}{}` après une césure explicite ou un tiret.

Exemple :

```
% An ordinary discretionary hyphen (equivalent to \-):
\discretionary{-}{}{}
% A place where TeX can break a line, but should not
% insert a space if the line isn't broken there, e.g.,
% after a dash:
\discretionary{}{}{}
% Accounts for German usage: 'flicken', but 'flik-
% ken':
German ''fli\discretionary{k-}{k}{ck}en''
```

`\hyphenation {` ⟨*mot*⟩ ␣ ... ␣ ⟨*mot*⟩ `}`

TEX garde un dictionnaire des exceptions de ses règles de césure. Chaque entrée du dictionnaire indique comment un mot particulier doit être coupé. La commande `\hyphenation` ajoute des mots au dictionnaire. Son argument est un suite de mots séparée par des blancs. Les lettres majuscules et minuscules sont équivalent. Les césures dans chaque mot indique les endroits où TEX peut couper ce mot. Un mot sans césure ne sera jamais coupé. Dans tous les cas, vous pouvez outrepasser le dictionnaire de césure en utilisant `\-` pour une occurrence particulière d'un mot. Vous devez produire toutes les formes grammaticales d'un mot que vous voulez que TEX coup, c'est-à-dire, le singulier et le pluriel.

Exemple :

```
\hyphenation{Gry-phon my-co-phagy}
\hyphenation{man-u-script man-u-scripts piz-za}
```

`\uchyph` [⟨*nombre*⟩ paramètre]

Une valeur positive de `\uchyph` (césure majuscule) permet découper des mots, comme des noms propre, qui commence par une lettre capital. Une valeur à zéro ou négative inhibe de telle coupure. Plain TEX mets `\uchyph` à 1, donc TEX essaye normalement de couper les mots qui commence par une lettre capital.

`\showhyphens {` ⟨*mot*⟩ ␣ ... ␣ ⟨*mot*⟩ `}`

Cette commande n'est habituellement pas utilisée dans des documents, mais vous pouvez l'utiliser sur votre terminal pour voir comment TEX

couperait certain jeu de mots aléatoires. Les mots, avec les coupures visibles, apparaissent dans la log et sur votre terminal. Vous obtiendrez une plainte à propos d'une "underfull hbox"—ignorez la.

Exemple :

```
\showhyphens{threshold quizzical draughts argumentative}
```

produit dans la log :

```
Underfull \hbox (badness 10000) detected at line 0
[] \tenrm thresh-old quizzi-cal draughts ar-gu-men-ta-tive
```

`\language` [⟨*number*⟩ paramètre]

Des langages différents ont différent jeux de règles de césure. Ce paramètre détermine le jeu de règles de césure que TEX utilise. En changeant `\language` vous pouvez obtenir que TEX coupe des portions de texte ou des documents entiers en accord avec les règles de césure appropriée à un langage particulier. Votre support local sur TEX vous dira si des jeux additionnels de règles de césure sont accessibles (derrière celui pour l'anglais) et quel sont les valeurs appropriées de `\language`. La valeur par défaut de `\language` est 0.

TEX mets le langage courant à 0 au début de tous les paragraphes, et compare `\language` au langage courant à chaque fois qu'il ajoute un caractère au paragraphe courant. Si ce n'est pas le même, TEX ajoute un élément extraordinaire indiquant que le langage change. Cet élément extraordinaire est la preuve pour les prochaines exécutions que les règles de langage peuvent changer.

`\setlanguage` ⟨*nombre*⟩

Cette commande mets le langage courant à ⟨*nombre*⟩ en insérant le même élément extraordinaire que vous obtiendrez en changeant `\language`. En outre, elle ne change pas la valeur de `\language`.

`\lefthyphenmin` [⟨*nombre*⟩ paramètre]
`\righthyphenmin` [⟨*nombre*⟩ paramètre]

Ces paramètres spécifient les plus petits fragments de mots que TEX autorise à gauche et à droite d'un mot coupé. Plain TEX les initialise par défaut à 2 et 3 respectivement ; ce sont les valeurs recommandées pour l'anglais.

`\hyphenchar` ⟨*police*⟩ [⟨*nombre*⟩ paramètre]

TEX n'utilise pas nécessairement le caractère '-' aux points de césure. à la place, il utilise le `\hyphenchar` de la police courante, qui est habituellement '-' mais pas nécessairement. Si une police a une valeur `\hyphenchar` négative, TEX ne coupe pas de mots dans cette police.

Notez que ⟨*police*⟩ est une séquence de contrôle qui nommes une police, pas un ⟨*nom de police*⟩ qui nommes des fichiers de police. Attention : un assignement à `\hyphenchar` n'est *pas* réinitialisé à la fin d'un groupe. Si vous voulez changer `\hyphenchar` localement, vous devrez sauvegarder et restaurer sa valeur originale explicitement.

Exemple :

```
\hyphenchar\tenrm = `-
   % Set hyphenation for tenrm font to `-'.
\hyphenchar\tentt = -1
   % Don't hyphenate words in font tentt.
```

`\defaulthyphenchar` [⟨*nombre*⟩ paramètre]

Quand TEX lit le fichier de métriques pour une police en réponse à une commande `\font`, il mets le `\hyphenchar` de la police dans `\defaulthyphenchar`. Si la valeur de `\defaulthyphenchar` n'est pas dans la fourchette 0–255 quand vous chargez une police, TEX ne coupera aucun mot dans cette police à moins que vous outrepassiez la décision en mettant le `\hyphenchar` de la police après. Plain TEX mets `\defaulthyphenchar` à 45, le code ASCII pour '-'.

Exemple :

```
\defaulthyphenchar = `-
   % Assume `-' is the hyphen, unless overridden.
\defaulthyphenchar = -1
   % Don't hyphenate, unless overridden.
```

Voir aussi : `\pretolerance` (p. 128).

Entêtes de section, listes et théorèmes

☞ `\beginsection` ⟨*argument*⟩ `\par`

Vous pouvez utiliser cette commande pour débuter une subdivision majeure de votre document. ⟨*argument*⟩ est prévu pour servir de titre de section. `\beginsection` amplifie ⟨*argument*⟩ par de l'espace vertical supplémentaire et le mets en police grasse, justifié à gauche. Vous pouvez produire le `\par` qui termine ⟨*argument*⟩ avec une ligne blanche.

Exemple :

```
$\ldots$  till she had brought herself down to nine
inches high.

\beginsection Section 6. Pig and Pepper

For a minute or two she stood looking at the house $\ldots$
```

produit :

... till she had brought herself down to nine inches high.

Section 6. Pig and Pepper

For a minute or two she stood looking at the house ...

`\item` ⟨*argument*⟩
`\itemitem` ⟨*argument*⟩

Cette commandes est utile pour créer des listes d'éléments. Tout le paragraphe suivant ⟨*argument*⟩ est indenté de `\parindent` (pour `\item`) ou par `2\parindent` (pour `\itemitem`). (Voir page 119 pour une explication de `\parindent`.) Ensuite ⟨*argument*⟩, suivi par un espace demi-cadratin, est placé juste à la gauche du texte de la première ligne du paragraphe pour qu'il tombe avec l'indentation du paragraphe comme spécifié par `\parindent`.

Si vous voulez inclure plus d'un paragraphe dans un élément, mettez `\item{}` devant les paragraphes additionnels.

Exemple :

```
{\parindent = 18pt
\noindent Here is what we require:
\item{1.}Three eggs in their shells,
but with the yolks removed.
\item{2.}Two separate glass cups containing:
\itemitem{(a)}One-half cup {\it used} motor oil.
\itemitem{(b)}One cup port wine, preferably French.
\item{3.}Juice and skin of one turnip.}
```

produit :

Here is what we require:

1. Three eggs in their shells, but with the yolks removed.
2. Two separate glass cups containing:
   (a) One-half cup *used* motor oil.
   (b) One cup port wine, preferably French.
3. Juice and skin of one turnip.

☞ `\proclaim` ⟨*argument*⟩`.␣`⟨*texte général*⟩`\par`

Cette commande "proclame" un théorème, un lemme, une hypothèse, etc. Elle mets ⟨*argument*⟩ en police grasse et le paragraphe suivant en italiques. ⟨*argument*⟩ doit être suivi par un point et un token espace,

qui servent à séparer ⟨*argument*⟩ du ⟨*texte général*⟩. ⟨*texte général*⟩ est constitué du texte jusqu'au frontières du paragraphe suivant, sauf que vous pouvez inclure des paragraphes multiples en les mettant entre accolades et terminer un paragraphe après l'accolade droite fermante.

Exemple :

```
\proclaim Theorem 1.
What I say is not to be believed.

\proclaim Corollary 1. Theorem 1 is false.\par
```

produit :

Theorem 1. *What I say is not to be believed.*

Corollary 1. *Theorem 1 is false.*

6 Commandes pour composer des pages

Cette section couvre des commandes qui concernent les pages, leurs composants et la routine de sortie. Pour une explication des conventions utilisées dans cette section, voir "Descriptions des commandes" (p. 3).

Espaces inter-ligne et inter-paragraphe

`\baselineskip` [⟨*ressort*⟩ paramètre]
`\lineskiplimit` [⟨*dimension*⟩ paramètre]
`\lineskip` [⟨*ressort*⟩ paramètre]

Ces trois paramètres déterminent ensemble combien d'espace TEX laisse entre des boîtes consécutive d'une liste verticale ordinaire, c'est-à-dire, les lignes d'un paragraphe. Cet espace est appelé "ressort inter-ligne". Il est aussi inséré entre les boîtes composant une vbox construite en mode vertical interne.

Danq le cas courant, quand les boîtes ne sont pas anormalement hautes ou profonde, TEX rend la distance de la ligne de base d'une boîte à la ligne de base de la suivante égal à `\baselineskip`. Il fait cela en insérant un ressort inter-ligne égal à `\baselineskip` moins la profondeur de la boîte supérieure (ainsi donné par `\prevdepth`) et la hauteur de la boîte du dessous. Mais si ce ressort inter-ligne doit être inférieur à `\lineskiplimit`, indiquant que les deux boîtes sont trop rapprochées, TEX insère le ressort `\lineskip` à la place.[1] Voir les pages 79–80 de *The TEXbook* et 99–99 de la traduction française pour une description précise.

[1] TEX débute normalement le début d'une liste verticale en mettant `\prevdepth` à −1000 pt et en testant `\prevdepth` avant *toutes* les boîtes. Si `\prevdepth` ≤ −1000 pt il ne doit insérer aucun ressort inter-ligne.

Notez que `\baselineskip` et `\lineskip` mesurent des *choses différentes* : La distance entre des lignes de base d'un coté et la distance entre le bas d'une boîte et le haut de la boîte suivante d'un autre coté. Voir la page 78 de *The TEXbook* et 99 de la traduction française pour d'autres détails. Le premier exemple ci-dessous montre les effets de `\lineskiplimit`.

Vous pouvez obtenir l'effet d'un double espacement en doublant la valeur de `\baselineskip` comme illustré dans le second exemple ci-dessous. Une modification de `\baselineskip` à n'importe quel endroit précédent la fin d'un paragraphe affecte le paragraphe entier.

Exemple :

```
\baselineskip = 11pt \lineskiplimit = 1pt
\lineskip = 2pt plus .5pt
Sometimes you'll need to typeset a paragraph that has
tall material, such as a mathematical formula,  embedded
within it.  An example of such a formula is $n \choose k$.
Note the extra space above and below this line as
compared with the other lines.
(If the formula didn't project below the line,
we'd only get extra space above the line.)
```

produit :

Sometimes you'll need to typeset a paragraph that has tall material, such as a mathematical formula, embedded within it. An example of such a formula is $\binom{n}{k}$. Note the extra space above and below this line as compared with the other lines. (If the formula didn't project below the line, we'd only get extra space above the line.)

Exemple :

```
\baselineskip = 2\baselineskip % Start double spacing.
```

`\prevdepth` [⟨*dimension*⟩ paramètre]

Quand TEX ajoute une boîte à une liste verticale, il règle `\prevdepth` à la largeur de cette boîte. TEX met `\prevdepth` à −10000 pt au début d'une liste verticale, indiquant que le ressort inter-ligne normal doit être supprimé.

`\normalbaselineskip` [⟨*ressort*⟩ paramètre]
`\normallineskiplimit` [⟨*dimension*⟩ paramètre]
`\normallineskip` [⟨*ressort*⟩ paramètre]
`\normalbaselines`

Ces trois paramètres contienne des valeurs pour `\baselineskip`, `\lineskip` et `\lineskiplimit` respectivement. La commande `\normalbaselines` mets `\baselineskip`, `\lineskip` et `\lineskiplimit` aux valeurs contenues dans les trois paramètres.

`\offinterlineskip`
Cette commande demande à TEX d'arrêter d'insérer des ressorts interligne à partir de maintenant. à moins que vous vouliez qu'il soit effectif pour le reste du document (ce que vous ne voulez probablement pas), vous devez l'englober dans un groupe avec le texte que vous voulez qu'il affecte. Son principal usage est de vous laisser faire l'espacement interligne vous-même, c'est-à-dire, en utilisant struts, sans interférence du ressort inter-ligne normal de TEX. `\offinterlineskip` est souvent utile quand vous construisez un alignement horizontal.

Exemple :

```
\def\entry#1:#2 {\strut\quad#1\quad&\quad#2\quad\cr}
\offinterlineskip \tabskip = 0pt \halign{%
\vrule\quad\hfil#\hfil\quad\vrule&
   \quad\hfil#\hfil\quad\vrule\cr
\noalign{\hrule}
\vphantom{\vrule height 2pt}&\cr \noalign{\hrule}
\entry \it Opera:\it Composer
\vphantom{\vrule height 2pt}&\cr \noalign{\hrule}
\vphantom{\vrule height 2pt}&\cr
\entry Fidelio:Beethoven
\entry Peter Grimes:Britten
\entry Don Giovanni:Mozart
\vphantom{\vrule height 2pt}&\cr \noalign{\hrule}}
```

produit :

Opera	*Composer*
Fidelio	Beethoven
Peter Grimes	Britten
Don Giovanni	Mozart

`\nointerlineskip`
Cette commande demande à TEX de ne pas insérer de ressort inter-ligne devant la prochaine ligne. Elle n'a aucun effet sur les lignes suivante.

`\openup` ⟨*dimension*⟩
Cette commande augmente `\baselineskip` de ⟨*dimension*⟩. Une commande `\openup` avent la fin d'un paragraphe affecte tout le paragraphe, donc vous ne devez pas utiliser `\openup` pour changer `\baselineskip` dans un paragraphe. `\openup` est plus utile pour composer des tables et des affichages mathématique—un petit espace supplémentaire entre des rangés les rend souvent plus lisibles.

Exemple :

```
Alice picked up the White King very gently, and lifted him
across more slowly than she had lifted the Queen; but before
she put him on the table, she thought she might well dust
him a little, he was so covered with ashes.
\openup .5\baselineskip % 1.5 linespacing.
```

produit :

Alice picked up the White King very gently, and lifted him across more slowly than she had lifted the Queen; but before she put him on the table, she thought she might well dust him a little, he was so covered with ashes.

Coupures de page

Encourager ou décourager des coupures de page

`\break`

Cette commande force une coupure de page. à moins que vous ne fassiez quelque chose pour compléter la page, Vous obtiendrez sûrement un "underfull vbox". `\break` peut aussi être utilisé en mode horizontal.

`\nobreak`

Cette commande empêche une coupure de page là où elle aurait du arriver autrement. `\nobreak` peut aussi être utilisé en mode horizontal.

`\allowbreak`

Cette commande demande à TEX d'autoriser une coupure de page là où elle ne pourrait normalement pas arriver. `\allowbreak` peut aussi être utilisé en mode horizontal.

`\penalty` ⟨*nombre*⟩

Cette commande produit un élément de pénalité. L'élément de pénalité rend TEX plus ou moins désireux de couper une page à l'endroit où cet élément arrive. Une pénalité négative, c'est-à-dire, un bonus, encourage une coupure de page ; Une pénalité positive décourage une coupure de page. Une pénalité de 10000 ou plus empêche toute coupure, tandis

qu'une pénalité de −10000 ou moins force une coupure. `\penalty` peut aussi être utilisé en mode horizontal.

Exemple :

```
\def\break{\penalty-10000 } % as in plain TeX
\def\nobreak{\penalty10000 } % as in plain TeX
\def\allowbreak{\penalty0 } % as in plain TeX
```

`\goodbreak`

Cette commande termine un paragraphe et indique aussi à TEX que c'est un bon endroit pour couper la page.

`\smallbreak`
`\medbreak`
`\bigbreak`

Ces commandes indique des endroits de plus en plus désirable pour TEX de couper une page. Elles demande aussi à TEX d'insérer un `\smallskip`, `\medskip` ou `\bigskip` (p. 160) si la coupure de page ne se produit pas là. TEX supprime ce saut s'il se produit juste après un saut égal ou plus grand.

☞ `\eject`
`\supereject`

Ces commandes force une coupure de page à la position courante et terminent le paragraphe courant. Si vous ne les précédez pas de `\vfil` (p. 163), TEX essayera d'étirer le contenu de la page (et se plaindra probablement d'un "underfull vbox"). La commande `\supereject`, en plus, enclenche la routine de sortie de plain TEX pour faire sortir toutes insertions en surplus, telles que de longues notes de pied de page, ainsi elle sont produites avant que tout autre entrée soit exécutée. Donc `\supereject` est un bonne commande à utiliser à la fin de chaque chapitre ou autre division majeure de votre document.

`\filbreak`

Cette commande procure une sorte de coupure de page conditionnelle. Elle demande à TEX de couper la page—sauf si le texte d'une autre `\filbreak` se trouve aussi sur la même page. En englobant un paragraphe dans une paire de `\filbreak`, vous pouvez vous assurer que TEX gardera un paragraphe sur une seule page s'il le peut. Vous ne devez pas utiliser `\filbreak` dans un paragraphe, car il force TEX en mode vertical et ainsi termine la paragraphe. Voir page 274 pour plus de conseil sur ce sujet.

`\raggedbottom`
`\normalbottom`
Normalement TEX essaye fortement de s'assurer que toutes les pages ait la même profondeur, c'est-à-dire, que leurs marges du bas soient égales. La commande `\raggedbottom` demande à TEX d'autoriser quelques variations parmi les marges du bas sur différentes pages. Il est souvent approprié d'utiliser `\raggedbottom` quand vous avez du matériel qui contient de large blocs de matériel qui ne doit pas être séparé sur deux pages. La commande `\normalbottom` efface l'effet de `\raggedbottom`.

Paramètres de coupure de page

`\interlinepenalty` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie la pénalité pour une coupure de page entre les lignes d'un paragraphe. En le mettant à 10000 vous pouvez forcer toutes les coupures de page à se faire entre les paragraphes, en espérant que les pages ait assez d'étirement pour que TEX puisse les composer décemment. Plain TEX laisse `\interlinepenalty` à 0.

`\clubpenalty` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie la pénalité pour une coupure de page juste après la première ligne d'un paragraphe. Une ligne seule en bas d'une page est appelée une "ligne orpheline". Plain TEX met `\clubpenalty` à 150.

`\widowpenalty` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie la pénalité pour une coupure de page juste avant la dernière ligne d'un paragraphe. Une ligne seule en haut d'une page est appelée une "ligne veuve". Plain TEX met `\widowpenalty` à 150.

`\displaywidowpenalty` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie la pénalité pour une coupure de page juste avant la dernière ligne d'une paragraphe partiel qui précède immédiatement un affichage mathématique. Plain TEX met `\displaywidowpenalty` à 50.

`\predisplaypenalty` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie la pénalité pour une coupure de page juste avant un affichage mathématique. Plain TEX met `\predisplaypenalty` à 10000.

`\postdisplaypenalty` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie la pénalité pour une coupure de page juste après un affichage mathématique. Plain TEX laisse `\postdisplaypenalty` à 0.

`\brokenpenalty` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie la pénalité pour une coupure de page juste après une ligne se terminant par une élément optionnel (habituellement une césure). `\brokenpenalty` s'applique aux coupure de page, tandis que `\hyphenpenalty` (p. 131) s'applique aux coupure de ligne. Plain TEX met `\brokenpenalty` à 100.

`\insertpenalties` [⟨*nombre*⟩ paramètre]
Ce paramètre contient la somme de certaines pénalités que TEX accumule quand il place des insertions sur la page courante. Ces pénalités encourent une plainte quand TEX exécute une commande `\insert` et découvre qu'un insertion précédente du même type sur cette page a été séparée, laissant une partie pour les page suivantes. Voir les pages 123–125 de *The TEXbook* et 999–999 de la traduction française pour les détails de ce calcul.

`\insertpenalties` a une signification entièrement différente pendant une routine de sortie—c'est le nombre d'insertions qui ont été vu mais qui ne tiennent pas sur la page courante (voir la page 125 de *The TEXbook* et 999 de la traduction française).

`\floatingpenalty` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifies la pénalitè que TEX ajoute à `\insertpenalties` quand le constructeur de page ajoute une insertion à la page courante et découvre qu'une insertion précédente du même type sur cette page a été coupé, en laissant une partie d'elle pour des pages suivantes. Plain TEX laisse `\floatingpenalty` à 0.

`\pagegoal` [⟨*dimension*⟩ paramètre]
Ce paramètre spécifie la hauteur désirée pour la page courante. TEX mets `\pagegoal` à la valeur courante de `\vsize` quand il met en premier une boîte ou une insertion sur la page courante. Vous pouvez raccourcir une page tant que TEX travaille sur elle en changeant la valeur de `\pagegoal`—même si la nouvelle valeur est inférieure à la hauteur du matériel déjà sur cette page. TEX mettra juste le matériel en plus sur la page suivante. Mais souvenez-vous—`\pagegoal` est remis à `\vsize` quand TEX commence la page suivante.

`\pagetotal` [⟨*dimension*⟩ paramètre]
Ce paramètre spécifie la hauteur naturelle accumulée sur la page courante. TEX met à jour `\pagetotal` quand il ajoute des éléments à la liste verticale principale.

`\pagedepth` [⟨*dimension*⟩ paramètre]
Ce paramètre spécifie la profondeur de la page courante. TEX met à jour `\pagedepth` quand il ajoute des éléments à la liste verticale principale.

`\pageshrink` [⟨*dimension*⟩ paramètre]
Ce paramètre spécifie le montant de rétrécissement dans les ressort accumulés sur la page courante. TEX met à jour `\pageshrink` quand il ajoute des éléments à la liste verticale principale.

`\pagestretch` [⟨*dimension*⟩ paramètre]
`\pagefilstretch` [⟨*dimension*⟩ paramètre]
`\pagefillstretch` [⟨*dimension*⟩ paramètre]
`\pagefilllstretch` [⟨*dimension*⟩ paramètre]
Ces quatre paramètres spécifie ensemble le montant d'étirement dans les ressort de la page courante. Le montant d'étirement a la forme $n_0 + n_1$`fil`$+n_2$`fill`$+n_3$`filll`, avec les quatre paramètres donnant les valeurs des quatre n_i. TEX met à jour ces paramètres quand il ajoute des éléments à la liste vertical principale.

Gabarit de page

Paramètres de description de page

`\hsize` [⟨*dimension*⟩ paramètre]
Ce paramètre spécifie la longueur de la ligne courante. Voir page 120 pour une plus complète explication.

`\vsize` [⟨*dimension*⟩ paramètre]
Ce paramètre spécifie l'extension verticale courante d'une page. TEX ne l'examine qu'en commençant une page. Donc si vous changez `\vsize` dans le milieu d'une page, votre changement n'affectera rien avant la page suivante. Si vous voulez changer l'extension verticale d'une page quand vous en êtes au milieu, vous pouvez assigner la nouvelle hauteur à `\pagegoal` (p. 145) à la place. (Si vous voulez que le changement affecte aussi les pages suivantes, vous devez changer *ensemble* `\vsize` et `\pagegoal`.) Plain TEX mets `\vsize` à `8.9in`.

`\hoffset` [⟨*dimension*⟩ paramètre]
`\voffset` [⟨*dimension*⟩ paramètre]
TEX prend normalement l'"origine" d'une page, qui est le point où il début l'impression, comme étant à un pouce sous le haut de la page et à un pouce à droite du bord gauche de la page.[2] Les valeurs de `\hoffset`

[2] TEX lui-même est indifférent d'où est l'origine de la page, mais cette information doit être inscrite pour les drivers du système qui convertissent les fichier `.dvi`en forme imprimable, ainsi ces différents systèmes donneront les même résultats.

et `\voffset` donnent l'offset horizontal et vertical de l'origine actuelle à partir de ce point. Ainsi si `\hoffset` et `\voffset` sont tous les deux à zéro, TEX utilise son origine normale.

Exemple :

```
\hoffset = -.3in
   % Start printing .7 inches from left edge of paper.
\voffset = 1in
   % Start printing 2 inches from top edge of paper.
```

`\topskip` [⟨*ressort*⟩ paramètre]

TEX insère un ressort en haut de chaque page pour s'assurer que la ligne de base de la première boîte de la page soit toujours à la même distance d du haut de la page. `\topskip` détermine le montant de ce ressort, appelé le "ressort `\topskip`", en spécifiant ce que d doit être (prévoyez que la première boîte de la page ne soit pas trop haute). d est donné par la taille naturelle du ressort `\topskip`. Si la hauteur de la première boîte sur la page dépasse d, et que le ressort devrait être négatif, TEX n'insère simplement pas de ressort `\topskip` du tout sur cette page.

Pour mieux comprendre l'effet de ces règles, supposez que `\topskip` n'a ni étirement ni rétrécissement et que le premier élément sur la page soit vraiment une boîte. Alors si la hauteur de cette boîte n'est pas plus grande que `\topskip`, sa ligne de base sera à `\topskip` du haut de la page indépendamment de sa hauteur. D'un autre coté, si la hauteur de la boîte est e, plus grand que `\topskip`, sa ligne de base sera à `\topskip` + e du haut de la page. Voir les pages 113–114 de *The TEXbook* et 999–999 de la traduction française pour les autres détails sur le fonctionnement de `\topskip`. Plain TEX mets `\topskip` à `10pt`.

`\parskip` [⟨*ressort*⟩ paramètre]

Ce paramètre spécifie le "saut de paragraphe", c'est-à-dire, le ressort vertical que TEX insère au début d'un paragraphe. Voir `\par` (p. 116) pour plus d'information sur ce qui se passe quand TEX débute un paragraphe. Plain TEX mets `\parskip` à `0pt plus 0.1pt`.

`\maxdepth` [⟨*dimension*⟩ paramètre]

Ce paramètre spécifie la profondeur maximum de la boîte du bas d'une page. Il est relié à `\boxmaxdepth` (p. 169). Si la profondeur de la boîte du bas d'une page dépasse `\maxdepth`, TEX descend le point de référence de la boîte pour qu'il soit à `\maxdepth` du bas de cette boîte. Sans cette ajustement, la boîte du bas d'une page pourrait bien s'étendre dans la marge du bas ou même recouvrir la page entièrement. Plain TEX mets `\maxdepth` à `4pt`.

Numéros de page

`\pageno` [⟨*nombre*⟩ paramètre]
Ce paramètre contient le numéro de page courant sous la forme d'un entier. Le numéro de page est normalement négatif pour les pages d'introduction qui sont numérotées avec des petits chiffres romains au lieu des nombres arabes. Si vous changez le numéro de page dans une page, le numéro modifié sera utilisé dans tous les entêtes ou pieds de page qui apparaissent sur cette page. L'impression actuelle des numéros de page est prise en main par la routine de sortie de TEX, que vous pouvez modifier.

Plain TEX garde le numéro de page dans le registre `\count0`. (`\pageno` est, en fait, un synonyme de `\count0`.) à chaque fois qu'il envoie une page vers le fichier `.dvi`, TEX affiche la valeur courante de `\count0` sur votre terminal pour que vous puissiez dire sur quelle page il est en train de travailler. Il est possible d'utiliser des registres `\count1`–`\count9` pour emboîter des niveaux de numéros de page (vous devez programmer cela vous-même). Si un de ces registres est différent de zéro, TEX l'affiche aussi sur votre terminal.[3]

Exemple :
```
This explanation appears on page \number\pageno\
of our book.
```
produit :
This explanation appears on page 148 of our book.

Exemple :
```
\pageno = 30 % Number the next page as 30.
Don't look for this explanation on page \number\pageno.
```
produit :
Don't look for this explanation on page 30.

`\advancepageno`
Cette commande ajoute 1 au numéro de page n dans `\pageno` si $n \geq 0$ et lui soustrait 1 si $n < 0$.

☞ `\nopagenumbers`
Par défaut, plain TEX produit un pied de page contenant un numéro de page centré. Cette commande demande à TEX de produire un pied de page blanc à la place.

[3] Plus précisément, il affiche tous les registres en séquence de `\count0` à `\count9`, mais omet les registres restants à zéro. Par exemple, si les valeurs de `\count0`–`\count3` sont $(17, 0, 0, 7)$ et que les autres sont à 0, TEX affiche le numéro de page ainsi `[17.0.0.7]`.

`\folio`
Cette commande produit le numéro de page courant, dont la valeur est le numéro n contenu dans `\pageno`. Si $n \geq 0$, TEX produit n comme un nombre décimal, tandis que si $n < 0$, TEX produit $-n$ en chiffre romain minuscule.

Exemple :
```
This explanation appears on page \folio\ of the book.
```
produit :

This explanation appears on page 149 of the book.

Lignes d'entête et de pied de page

`\headline` [⟨*liste de token*⟩ paramètre]
`\footline` [⟨*liste de token*⟩ paramètre]

Ces paramètres contiennent, respectivement, la ligne de tête courante (entête) et la ligne de pied de page courante (pied de page). La routine de sortie de plain TEX place la ligne de tête en haut de chaque page et ligne de pied de page en bas de chaque page. La ligne de tête par défaut est vide et la ligne de pied de page par défaut est un numéro de page centré.

Les lignes de tête et de pied de page doivent être toutes les deux aussi large que `\hsize` (utilisez `\hfil`, p. 163, pour cela si nécessaire). Vous devez toujours inclure une commande de sélection de police dans ces lignes, sinon la police courante est imprévisible quand TEX appelle la routine de sortie. Si vous ne sélectionnez la police explicitement, vous obtiendrez n'importe quelle police que TEX utilisait quand il a coupé la page.

N'essayez pas d'utiliser `\headline` ou `\footline` pour produire des entêtes ou des pieds de page multi-lignes. Bien que TEX ne se plaindra pas, cela vous donnera quelque chose de très laid. Voir page 282 pour une méthode de création d'entête ou de pied de page multi-ligne.

Exemple :
```
\headline = {\tenrm My First Reader\hfil Page \folio}
```
produit :

My First Reader Page 10
(*at the top of page 10*)

Exemple :

```
\footline = {\tenit\ifodd\pageno\hfil\folio
           \else\folio\hfil\fi}
% Produce the page number in ten-point italic at
% the outside bottom corner of each page.
```

Marques

`\mark {` ⟨*texte*⟩ `}`
Cette commande fait que TEX attache une marque contenant ⟨*mark text*⟩ à toute liste qu'il est en train de construire. Généralement vous ne devez pas utiliser `\mark` dans une construction "interne" comme une formule mathématique ou une boîte que vous avez construit avec une commande `\hbox`, `\vbox` ou `\vtop`, parce que TEX ne verra pas la marque quand il construira la boîte principale de la page. Mais si vous utilisez `\mark` en mode horizontal ordinaire ou directement dans une hbox qu fait partie de la liste vertical principale, la marque migrera vers la liste vertical principale. Voir les pages 259–260 de *The TEXbook* et 999-999 de la traduction française pour des exemples montrant comment `\mark` peut être utilisé.

`\firstmark`
`\botmark`
`\topmark`
Ces commandes développent le texte de la marque dans un élément généré par une précédente commande `\mark`. Le texte de la marque a la forme d'une liste de token. TEX mets les valeurs de ces commandes quand il termine le placement du contenu d'une page dans `\box255`, juste avant d'appeler la routine de sortie comme une partie de son action de coupure de page. TEX détermine ces valeurs comme suit :

- `\firstmark` contient les tokens de la première marque de la page.
- `\botmark` contient les tokens de la dernière marque de la page.
- `\topmark` contient les tokens de la marque qui est effective tout en haut de la page. Cette marque est la dernière marque qui *précède* la page, c'est-à-dire, le `\botmark` de la page précédente. Il est vide si aucune page ne précède la page.

Si une page n'a aucune marque sur elle, TEX mettra `\firstmark` et `\botmark` avec la même marque que `\topmark`, c'est-à-dire, la plus récente marque précédente. La table en bas de la page 258 de *The TEXbook* et 999 de la traduction française illustre la relation entre `\firstmark`, `\botmark` et `\topmark`.

`\splitfirstmark`
`\splitbotmark`
Ces commandes développent le texte de marque généré par une commande `\mark` précédente qui produisent un élément dans la liste d'élément d'une

vbox V. Le texte de marque a la forme d'une liste de token. Quand TEX sépare V en réponse à une commande `\vsplit` (p. 155), il mets les valeurs de ces commandes comme suit :

- `\splitfirstmark` contient les tokens de la première marque dans la liste d'élément de V.
- `\splitbotmark` contient les tokens de la dernière marque dans la liste d'élément de V.

Ces commandes ne produisent aucun token s'il n'y a pas de `\vsplit` précédent ou si le plus récent `\vsplit` précédent ne contient aucune marque.

Insertions

Pieds de page

☞ `\footnote` ⟨*argument*$_1$⟩ ⟨*argument*$_2$⟩
`\vfootnote` ⟨*argument*$_1$⟩ ⟨*argument*$_2$⟩

Ces commandes produisent des notes de pied de page. ⟨*argument*$_1$⟩ est la "marque de référence" pour la note de pied de page et ⟨*argument*$_2$⟩ est son texte. Le texte peut faire plusieurs paragraphes de long si nécessaire et peut contenir des constructions comme des affichages mathématiques, mais ne peut contenir aucune insertions (comme d'autres notes de pied de page).

Vous ne pouvez pas utiliser ces commandes à l'intérieur d'une sous-formule d'une formule mathématique, dans une boîte à l'intérieur d'une boîte contribuant à une page ou dans une insertion de toute sorte. Si vous n'êtes pas sur de la manière dont ces restrictions s'applique, il peut être sage de n'utiliser `\footnote` et `\vfootnote` que directement dans un paragraphe ou entre des paragraphes.

Ces restrictions ne sont pas aussi sévère qu'elles ne le semblent parce que vous pouvez utiliser `\vfootnote` pour mettre en note de pied de page la plupart des choses. `\footnote` et `\vfootnote` insèrent tous les deux des marques de référence en tête de la note de pied de page elle-même, mais `\vfootnote` n'insère pas la marque de référence dans le texte. Ainsi, quand vous utilisez `\vfootnote` vous pouvez insérer explicitement la marque de référence n'importe où, sans que cela concerne le contexte et placer le `\vfootnote` dans le paragraphe suivant. Si vous trouvez que la note de bas de page atterrit sur la page suivant celle où elle devrait, déplacez le `\vfootnote` vers le paragraphe précédent. Il y a de rare circonstances où vous devez altérer le texte de votre document

pour qu'une note de bas de page apparaisse sur la même page que sa marque de référence.

Exemple :

```
To quote the mathematician P\'olya is a ploy.\footnote
*{This is an example of an anagram, but not a strict one.}
```

produit :

To quote the mathematician Pólya is a ploy.*

⋮

* This is an example of an anagram, but not a strict one.

Exemple :

```
$$f(t)=\sigma\sigma t\;\raise 1ex \hbox{\dag}$$
\vfootnote \dag{The $\sigma\sigma$ notation was explained in
the previous section.}
```

produit :

$$f(t) = \sigma\sigma t\,^{\dagger}$$

⋮

† The $\sigma\sigma$ notation was explained in the previous section.

Insertions générales

`\topinsert` ⟨*matériel en mode vertical*⟩ `\endinsert`
`\midinsert` ⟨*matériel en mode vertical*⟩ `\endinsert`
`\pageinsert` ⟨*matériel en mode vertical*⟩ `\endinsert`

Ces commandes produisent différentes formes d'insertions qui informent (ou autorisent) TEX à délocaliser le ⟨*matériel en mode vertical*⟩ :

- `\topinsert` essaye de mettre le matériel en haut de la page courante. S'il ne peut pas s'y ajuster, `\topinsert` déplacera le matériel vers le prochain haut de page possible.
- `\midinsert` essaye de mettre le matériel à la position courante. S'il ne peut pas s'y ajuster, `\midinsert` déplacera le matériel vers le prochain haut de page possible.
- `\pageinsert` mets le matériel lui-même sur la page suivante. Pour éviter une page pas assez pleine, assurez vous de terminer le matériel inséré avec `\vfil` ou remplissez l'espace excessif d'une autre manière.

Le ⟨*matériel en mode vertical*⟩ est dit être "flottant" parce que TEX peut le déplacer d'une place à une autre. Des insertions sont très pratique pour du matériel tel que des figures et des tables parce que vous pouvez positionner ce matériel où vous voulez sans savoir où les coupures de page tomberont.

Chacune de ces commandes terminent implicitement le paragraphe courant, donc vous ne devez les utiliser qu'entre des paragraphes. Vous ne devez pas les utiliser dans une boîte ou dans d'autres insertions. Si vous avez plusieurs insertions en compétition pour le même espace, TEX conservera leur ordre relatif.

Exemple :

```
\pageinsert
% This text will appear on the following page, by itself.
This page is reserved for a picture of the Queen of Hearts
sharing a plate of oysters with the Walrus and
the Carpenter.
\endinsert
```

`\endinsert`

Cette commande termine une insertion débutée par `\topinsert`, `\midinsert` ou `\pageinsert`.

`\insert` ⟨*nombre*⟩ `{` ⟨*matériel en mode vertical*⟩ `}`

Cette commande primitive fournit le mécanisme sous-jacent la construction des insertions, mais elle n'est presque jamais utilisée en dehors d'un définition de macro. Les définitions des commandes `\footnote`, `\vfootnote`, `\topinsert`, `\midinsert` et `\pageinsert` sont toutes construite autour d'`\insert`.

Quand vous concevez des insertions pour un document, vous devez assigner un différent code entier[4] n pour chaque type d'insertion, utilisez la commande `\newinsert` (p. 252) pour obtenir les codes entiers. La commande `\insert` elle-même annexe le ⟨*matériel en mode vertical*⟩ à la liste verticale ou horizontale courante. Notre routine de sortie est responsable de déplacer le matériel inséré d'où il réside dans la `\box`n vers la page de sortie.

TEX regroupe ensemble toutes les insertions ayant le même numéro de code. Chaque code d'insertion n a quatre registres qui lui sont associés :

- `\box`n est l'endroit où TEX accumule le matériel pour des insertions avec le code n. Quand TEX coupe une page, il mets dans `\box`n autant de matériel d'insertion n qu'il pourra sur la page. Notre routine de sortie devra alors déplacer ce matériel vers la page actuelle. Vous pouvez utiliser `\ifvoid` (p. 246) pour tester s'il y a du matériel dans `\box`n. Si tout le matériel ne rentre pas, TEX sauve le surplus pour la page suivante.
- `\count`n est un facteur de magnification f. Quand TEX doit traiter l'espace vertical occupé sur la page dans le matériel d'insertion n, il multiplie l'extension verticale de ce matériel par $f/1000$. Ainsi vous

[4] *The TEXbook* utilise le terme "classe" pour un code. Nous préférons un terme différent pour éviter toute confusion avec l'autre signification de "classe" (p. 56).

mettrez normalement f à 500 pour une insertion en double-colonne et à 0 pour une note marginale.

- `\dimen`n spécifie le montant maximum de matériel d'insertion n que TEX mettra sur une seule page.
- `\skip`n spécifie un espace supplémentaire que TEX alloue sur la page si la page contient du matériel d'insertion n. Cet espace est en plus de l'espace occupé par l'insertion elle-même. Par exemple, il justifiera l'espace sur la page au-dessus des notes de pied de page (s'il y en a).

TEX déclare `\box`n et vous devez déclarer les trois autres registres pour que TEX puisse calculer correctement l'espace vertical requis par l'insertion. Voir les pages 122–125 de *The TEXbook* et 999–999 de la traduction française Pour d'autres détails sur la façon dont TEX exécute ces commandes et comment des insertions interagissent avec les coupures de page.

Voir aussi : `\floatingpenalty` (p. 145).

Modifier la routine de sortie

`\output` [⟨*liste de token*⟩ paramètre]

Ce paramètre contient la routine de sortie courante, c'est-à-dire, La liste de token que TEX développe quand il trouve une coupure de page. TEX mets la page dans la `\box255`, ainsi `\output` est responsable de faire quelque chose avec la `\box255`—soit la sortir, soit la mettre ailleurs. La routine de sortie est aussi responsable d'attacher des choses comme des entête ou des pieds de page.

`\plainoutput`

Cette command invoque la routine de sortie de plain TEX. Plain TEX définit `\output` comme une liste de token contenant le seule et unique token `\plainoutput`.

`\shipout` ⟨*boîte*⟩

Cette commande charge TEX d'envoyer ⟨*boîte*⟩ vers le fichier `.dvi`. TEX développe toute commande `\write` dans ⟨*boîte*⟩ comme une partie de `\shipout`. Le principal usage de `\shipout` est dans la routine de sortie, mais vous pouvez l'utiliser n'importe où.

`\deadcycles` [⟨*nombre*⟩ paramètre]
Ce paramètre contient le nombre de fois que TEX a lancé la routine de sortie depuis la dernière fois qu'il a fait un `\shipout`.[5] Si `\deadcycles` devient trop grand, TEX est probablement entré dans une boucle, par exemple, une où le constructeur de page essaye la même coupure de page à n'en plus finir.

`\maxdeadcycles` [⟨*nombre*⟩ paramètre]
Si la valeur de `\deadcycles` dépasse la valeur de `\maxdeadcycles`, TEX présume que la routine de sortie est entrée dans une boucle. TEX se plaint alors et effectue sa propre et simple routine de sortie, équivalente à `\shipout\box255`, Ce qui brisera vraisemblablement la boucle. Plain TEX mets `\maxdeadcycles` à 25.

`\outputpenalty` [⟨*nombre*⟩ paramètre]
TEX mets ce paramètre quand il coupe une page. Si le point de coupure était sur un élément de pénalité, TEX enlève l'élément de pénalité et mets `\outputpenalty` à la valeur de la pénalité au point de coupure ; autrement, il mets `\outputpenalty` à 0.

Supposez que vous défaites une coupure de page pour couper la page à un endroit différent que celui que TEX a choisi. Pour reconstruire la page, vous devez recréer la pénalité au point de coupure choisi par TEX. Vous pouvez accomplir ceci avec la commande `\penalty\outputpenalty`.

`\holdinginserts` [⟨*nombre*⟩ paramètre]
Si ce paramètre est plus grand que 0 quand TEX exécute une coupure de page, TEX s'abstiendra à exécuter des insertions. Mettre ce paramètre à 1 peut être utile quand vous écrivez une routine de sortie que à besoin de refaire le contenu de la page, par exemple, une routine de sortie qui utilise une valeur de `\vsize` (p. 146) différente de celle utilisée par le constructeur de page.

Séparer des listes verticales

`\vsplit` ⟨*nombre*⟩ `to` ⟨*dimension*⟩
Cette commande fait que TEX sépare la boîte numéro ⟨*number*⟩, que nous appellerons B_2, en deux parties. Il utilise le même algorithme que celui qu'il utiliserait si B_2 était une page et qu'il coupait cette page ; Le point de division alors correspond à la coupure de page qu'il trouverait. La boîte B_2 doit être une vbox, pas une hbox. TEX mets le matériel précédant

[5] Plus précisément, TEX mets `\deadcyles` à 0 à chaque fois qu'il exécute `\shipout` et l'incrémente de 1 à chaque fois qu'il exécute `\output`.

le point de division dans un autre boîte B_1 et laisse le matériel après le point de division dans B_2. La commande `\vsplit` alors produit B_1. Normalement vous assignerez B_1 à un registre de boîte différent, comme dans l'exemple ci-dessous. Si le point de division est à la fin de B_2, B_2 sera vide après le `\vsplit`.

TEX emploie son algorithme de coupure de page habituel pour la division. Il utilise ⟨*dimension*⟩ pour `\pagegoal`, la hauteur désirée de B_1. L'extension verticale de B_1 peut ne pas être exactement égale à ⟨*dimension*⟩ parce que TEX peut ne pas être capable de achever son but de page parfaitement. TEX ne prend pas en compte les insertions dans le calcul de la division, donc des insertions dans la liste verticale originale de B_2 seront maintenues mais n'affecteront pas le point de division.

Exemple :

```
\setbox 20 = \vsplit 30 to 7in
% Split off the first seven inches or so of material from
% box 30 and place that material in box 20.
```

`\splitmaxdepth` [⟨*dimension*⟩ paramètre]

Ce paramètre spécifie la profondeur maximum autorisée pour une boîte résultant d'un `\vsplit`. `\splitmaxdepth` joue le même rôle que celui de `\maxdepth` (p. 147) pour une page.

`\splittopskip` [⟨*ressort*⟩ paramètre]

Ce paramètre spécifie le ressort que TEX insère en haut d'une boîte résultant d'un `\vsplit`. `\splittopskip` joue le même rôle que `\topskip` (p. 147) joue pour une page.

Voir aussi : `\splitbotmark`, `\splitfirstmark` (p. 150).

7 Commandes pour les modes horizontaux et verticaux

Cette section couvre les commandes qui ont des formes correspondantes ou identiques dans les modes horizontaux et verticaux. Ces commandes fournissent des boîtes, des espaces, des filets, des réglures et des alignements. Pour une explication des conventions utilisées dans cette section, voir "Descriptions des commandes" (p. 3).

Produire des espaces

Espaces horizontaux de largeur fixe

`\thinspace`
Cette commande produit un crénage positif dont la largeur équivaut à un sixième d'em (p. 62) c'est-à-dire qu'il demande à TEX de déplacer sa position à droite de ce montant. c'est pratique quand vous avez par exemple des quotes imbriquées et que vous voulez les séparer. TEX ne coupe pas de ligne sur un `\thinspace`.

Exemple :

```
``\thinspace`A quote.'\thinspace''\par
24,\thinspace 29--31,\thinspace 45,\thinspace 102
```

produit :

"`A quote.'"
24, 29–31, 45, 102

`\negthinspace`
Cette commande produit un crénage négatif dont la largeur équivaut à un sixième d'em (p. 62) c'est-à-dire qu'il demande à TEX de déplacer sa

position à gauche de ce montant. C'est pratique pour rapprocher deux caractères qui sont un peu trop éloignés l'un de l'autre. TEX ne coupe pas de ligne sur un `\negthinspace`.

Exemple :

```
The horror, the horror\negthinspace, the horror of it all!
```

produit :

The horror, the horror, the horror of it all!

`\enspace`

Cette commande produit un crénage dont la largeur équivaut à un en (la moitié d'un em, voir page 62). TEX ne coupe pas de ligne sur un `\enspace` à moins qu'il soit suivi d'un ressort. Dans une liste à puce, les puces sont généralement séparées du texte qui les suit par un `\enspace`.

Exemple :

```
Lemma 1.\enspace There exists a white rabbit.
```

produit :

Lemma 1. There exists a white rabbit.

☞ `\enskip`
`\quad`
`\qquad`

Chacune de ces commandes produit un bout de ressort horizontal qui ne peut ni s'étirer ni se rétrécir. TEX peut couper une ligne sur un tel ressort. Les largeurs de ces ressort (qui sont relatifs à la police courante) sont les suivantes pour `cmr10`, la police par défaut de plain TEX :

Commande	*Espace*	*Illustration*
`\enskip`	1/2 em	→| |←
`\quad`	1 em	→| |←
`\qquad`	2 em	→| |←

Exemple :

```
en\enskip skip; quad\quad skip; qquad\qquad skip
```

produit :

en skip; quad skip; qquad skip

Espaces verticaux de largeur fixe

☞ `\smallskip`
`\medskip`
`\bigskip`

Ces commandes produisent des montants d'espacement vertical successivement plus large :

smallskip

medskip

bigskip

`\smallskip` saute de 3 points et peut s'étirer ou s'élargir de 1 point. `\medskip` est équivalent à deux `\smallskip`s et `\bigskip` est équivalent à deux `\medskip`s.

Ces commandes terminent un paragraphe car elles sont fondamentalement verticales. Les sauts qu'elles produisent sont ajoutés au saut interparagraphe normal.

Exemple :

```
Hop \smallskip skip \medskip and \bigskip jump.
```

produit :

Hop

skip

and

jump.

`\smallskipamount` [⟨*ressort*⟩ paramètre]
`\medskipamount` [⟨*ressort*⟩ paramètre]
`\bigskipamount` [⟨*ressort*⟩ paramètre]

Ces paramètres spécifient les montants de ressort produit par les commandes `\smallskip`, `\medskip` et `\bigskip`. En changeant ces paramètres vous changez l'effet des commandes. Les valeurs par défaut (pour plain TEX) correspondent à un quart d'une ligne d'espace, à la moitié d'une ligne d'espace, et à une ligne d'espace entière. Nous vous recommandons de maintenir ce ratio en changeant ces valeurs à chaque fois que vous changez `\baselineskip` (p. 139).

Espace de taille variable

☞ `\hskip` ⟨*dimension*$_1$⟩ `plus` ⟨*dimension*$_2$⟩ `minus` ⟨*dimension*$_3$⟩
`\vskip` ⟨*dimension*$_1$⟩ `plus` ⟨*dimension*$_2$⟩ `minus` ⟨*dimension*$_3$⟩

Ces commandes produisent des ressorts respectivement horizontaux et verticaux. Dans le cas le plus simple et le plus courant, quand seul ⟨*dimension*$_1$⟩ est présent, `\hskip` saute vers la droite de ⟨*dimension*$_1$⟩ et `\vskip` saute vers le bas de la page de ⟨*dimension*$_1$⟩. Plus généralement, ces commandes produisent des ressort dont la taille naturelle est ⟨*dimension*$_1$⟩, l'étirement est ⟨*dimension*$_2$⟩ et le rétrécissement ⟨*dimension*$_3$⟩. Le `plus` ⟨*dimension*$_2$⟩, le `minus` ⟨*dimension*$_3$⟩, ou les deux peuvent être omis. Si les deux sont présent, le `plus` doit être avant le `minus`. Une valeur omise est considérée à zéro. Toutes les ⟨*dimension*⟩s peuvent être négatives.

Vous pouvez utiliser `\hskip` en mode mathématique, mais vous ne pouvez utiliser d'unités `mu` (voir "unité mathématique", p. 100) pour aucune

des dimensions. Si vous voulez des unités `mu`, utilisez `\mskip` (p. 223) à la place.

Exemple :

```
\hbox to 2in{one\hskip 0pt plus .5in two}
```

produit :

one two

Exemple :

```
\hbox to 2in{Help me! I can't fit
{\hskip 0pt minus 2in} inside this box!}
```

produit :

Help me! I can't fitnside this box!

Exemple :

```
\vbox to 4pc{\offinterlineskip% Just show effects of \vskip.
   \hbox{one}\vskip 0pc plus 1pc \hbox{two}
      \vskip .5pc \hbox{three}}
```

produit :

one

two

three

`\hglue` ⟨*ressort*⟩
`\vglue` ⟨*ressort*⟩

La commande `\hglue` produit un ressort horizontal qui ne disparaît pas sur une coupure de ligne ; La commande `\vglue` produit un ressort vertical qui ne disparaît pas sur une coupure de page. Pour le reste ces commandes sont comme `\hskip` et `\vskip`. Vous pouvez utiliser `\vglue` pour produire de l'espace blanc en haut d'une page, par exemple, au dessus d'un titre sur la première page d'un document, mais `\topglue` (suivant) est normalement plus approprié pour cet usage.

`\topglue` ⟨*ressort*⟩

Cette commande[1] fait que l'espace entre le haut de la page et le haut de la première boîte de la page soit précisément égal à ⟨*glue*⟩. Le haut de la page est considéré être sur la ligne de base d'une ligne imaginaire de texte juste au-dessus de la ligne du haut de la page. Plus précisément, c'est un distance `\topskip` au dessus de l'origine donnée par `\hoffset` et `\voffset`.

[1] `\topglue` a été ajouté à TEX dans la version 3.0, après les autres aménagements introduit par le nouveau TEX (p. 18). C'est la première décrite dans la *dix-huitième* édition de *The TEXbook*.

Cette commande est pratique parce que TEX ajuste le ressort normalement produit par `\topskip` d'une manière complexe. En utilisant `\topglue` vous pouvez contrôler la position de la première boîte de la page sans vous inquiéter de ces ajustements.

`\kern` ⟨*dimension*⟩
L'effet de cette commande dépend du mode dans lequel est TEX quand il ma rencontre :

- En mode horizontal, TEX déplace sa position vers la droite (pour un crénage positif) ou vers la gauche (pour un crénage négatif).
- En mode vertical, TEX déplace sa position vers le bas de la page (pour un crénage positif) ou vers le haut de la page (pour un crénage négatif).

Ainsi un crénage positif produit un espace vide tandis qu'un crénage négatif fait que TEX recule sur quelque chose qu'il a déjà produit. Cette notion de crénage est différente de la notion de crénage dans certain système de composition typographique informatisés—dans TEX, des crénages positifs *écartent* deux lettres au lieu de les rapprocher.

Un crénage est similaire à un ressort, sauf que (a) un crénage ne peut ni s'étirer ni se rétrécir, et (b) TEX ne coupera une ligne ou une page sur un crénage que si le crénage est suivi par un ressort et ne fait pas partie d'une formule mathématique. Su TEX trouve un crénage à la fin d'une ligne ou d'une page, il élimine le crénage. Si vous voulez obtenir l'effet d'un crénage qui ne disparaît jamais, utilisez `\hglue` ou `\vglue`.

Vous pouvez utiliser `\kern` en mode mathématique, mais vous ne pouvez utiliser d'unités `mu` (voir "unité mathématique", p. 100) pour ⟨*dimension*⟩. Si vous voulez des unités `mu`, utilisez `\mkern` (p. 223) à la place.

Exemple :

```
\centerline{$\Downarrow$}\kern 3pt % a vertical kern
\centerline{$\Longrightarrow$\kern 6pt % a horizontal kern
   {\bf Heed my warning!}\kern 6pt % another horizontal kern
   $\Longleftarrow$}
\kern 3pt % another vertical kern
\centerline{$\Uparrow$}
```

produit :

$\Downarrow$

$\Longrightarrow$ **Heed my warning!** $\Longleftarrow$

$\Uparrow$

`\hfil`	`\vfil`
`\hfill`	`\vfill`

Ces commandes produisent des ressorts horizontaux et verticaux étirables infiniment qui écrasent tout étirement fini qui peut être présent. `\hfil` et

`\hfill` produisent des ressorts horizontaux, tandis que `\vfil` et `\vfill` produisent des ressorts verticaux.

`\hfill` est infiniment plus grand que `\hfil`. Si `\hfill` et `\hfil` apparaissent dans la même boîte, le `\hfill` consumera toute la place supplémentaire possible et le `\hfil` sera effectivement ignoré. `\hfill` peut à son tour est écrasé par `\hskip 0pt plus 1filll`. les ressort produits par `\hfil` et `\hfill` ne rétrécissent jamais.

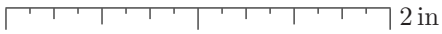
Le comportement de `\vfil` et `\vfill` est analogue.

Exemple :

```
\hbox to 2in{Left\hfil Middle \hfil Right}
```

produit :

Left Middle Right



Exemple :

```
\hbox to 2in{Left\hfil Middle \hfill Right}
```

produit :

LeftMiddle Right

Exemple :

```
\leftline{%
\vbox to 4pc{%
   \hbox{Top}\vfil\hbox{Middle}\vfil \hbox{Bottom}}\quad
\vbox to 4pc{%
   \hbox{Top}\vfil\hbox{Middle}\vfill\hbox{Bottom}}}
```

produit :

Top Top
Middle
Middle

Bottom Bottom

`\hss`
`\vss`

Ces commandes produisent des ressorts horizontaux et verticaux qui sont infiniment étirables et rétrécissables. Le ressort peut se rétrécir sur une distance négative, produisant l'effet de reculer sur une ligne (pour `\hss`) ou remonter sur une page (pour `\vss`).

Exemple :

```
\line{text\hfil\hbox to 0pt{margin\hss}}
% 'margin\hss' shrinks to the zero width of the hbox.
```

produit :

text margin

Exemple :

```
\vbox to 1pc{\hrule width 6pc % Top of box.
   \hbox{1} \vskip 1pc\hbox to 2pc{\hfil 2}
   % The \vss absorbs the extra distance produced by \vskip.
   \vss \hbox to 3pc{\hfil 3}
   \hrule width 6pc}% Bottom of box.
```

produit :

1 3

2

`\hfilneg`
`\vfilneg`

Ces commandes effacent l'effet d'un `\hfil` ou d'un `\vfil` précédent. Tandis que `\hfil` et `\vfil` produisent un ressort positif infiniment étirable, `\hfilneg` et `\vfilneg` produisent un ressort négatif infiniment étirable. (ainsi, n `\hfilneg` effacent n `\hfil` et pareillement pour le `\vfilneg`.) L'utilisation principale de `\hfilneg` et de `\vfilneg` est de contrecarrer l'effet d'un `\hfil` ou d'un `\vfil` inséré par une macro.

`\hfilneg` et `\vfilneg` ont la propriété curieuse, s'ils sont les seuls ressorts infiniment étirables d'une boîte, de produire exactement le même effet que `\hfil` et `\vfil`.

Exemple :

```
\leftline{\hfil on the right\hfilneg}
% Cancel the \hfil that \leftline produces to the right
% of its argument.
```

produit :

on the right

Exemple :

```
\def\a{\hbox to 1pc{\hfil 2}\vfil}
\vbox to 4pc{\hbox{1} \vfil \a
   \vfilneg \hbox to 2pc{\hfil 3}}
```

produit :

1

2

3

Voir aussi : `\hbadness` et `\vbadness` (p. 176), `\hfuzz` et `\vfuzz` (p. 176), "filets" (p. 69).

Manipuler des boîtes

Construire des hbox et des vbox

`\hbox {` ⟨*matériel en mode horizontal*⟩ `}`
`\hbox to` ⟨*dimension*⟩ `{` ⟨*matériel en mode horizontal*⟩ `}`
`\hbox spread` ⟨*dimension*⟩ `{` ⟨*matériel en mode horizontal*⟩ `}`

Ces commandes produisent une hbox (boîte horizontale) contenant du ⟨*matériel en mode horizontal*⟩. Les accolades autour du ⟨*matériel en mode horizontal*⟩ définissent un groupe. TEX ne coupe pas le ⟨*matériel en mode horizontal*⟩ en plusieurs lignes, puisqu'il est en mode horizontal restreint quand il assemble la boîte. TEX ne change pas la taille de la boîte une fois qu'elle est produite.

`\hbox` est souvent utile quand vous voulez conserver du texte sur une seule ligne. Si votre utilisation de `\hbox` empêche TEX de couper des lignes de manière acceptable, TEX se plaindra d'un "overfull hbox".

La largeur du hbox dépend des arguments de `\hbox` :

- Si vous ne spécifiez que du ⟨*matériel en mode horizontal*⟩, le hbox aura sa largeur naturelle.
- Si vous spécifiez `to` ⟨*dimension*⟩, la largeur du hbox sera ⟨*dimension*⟩.
- Si vous spécifiez `spread` ⟨*dimension*⟩, la largeur du hbox sera sa largeur naturelle plus ⟨*dimension*⟩, c'est-à-dire, le hbox s'allongera de ⟨*dimension*⟩.

La commande `\hfil` (p. 163) est utile pour remplir un hbox d'espace vide quand le matériel dan la boîte n'est pas aussi large que la largeur de la boîte.

Exemple :

```
\hbox{ugly suburban sprawl}
\hbox to 2in{ugly \hfil suburban \hfil sprawl}
\hbox spread 1in {ugly \hfil suburban \hfil sprawl}
% Without \hfil in the two preceding lines,
% you'd get 'underfull hbox'es.
```

produit :

ugly suburban sprawl
ugly suburban sprawl
ugly suburban sprawl

`\vtop` ⟨*matériel en mode vertical*⟩
`\vtop to` ⟨*dimension*⟩ `{` ⟨*matériel en mode vertical*⟩ `}`
`\vtop spread` ⟨*dimension*⟩ `{` ⟨*matériel en mode vertical*⟩ `}`
`\vbox {` ⟨*matériel en mode vertical*⟩ `}`
`\vbox to` ⟨*dimension*⟩ `{` ⟨*matériel en mode vertical*⟩ `}`
`\vbox spread` ⟨*dimension*⟩ `{` ⟨*matériel en mode vertical*⟩ `}`

Ces commandes produisent une vbox (boîte verticale) contenant du ⟨*matériel en mode vertical*⟩. Les accolades autour du ⟨*matériel en mode vertical*⟩ définissent un groupe. TEX est en mode vertical interne quand il assemble la boîte. TEX ne change pas la taille de la boîte un fois qu'elle est produite.

La différence entre `\vtop` et `\vbox` consiste en où TEX met le point de référence de la vbox construite. Normalement, le point de référence obtenu de `\vtop` tend à être sur ou près du haut de la vbox construite, tandis que le point de référence obtenu de `\vbox` tend à être sur ou près du bas de la vbox construite. Ainsi une rangée de vbox toutes construites avec `\vtop` tendra à avoir leurs hauts à peu près alignés, tandis qu'une rangé de vbox toutes construites avec `\vbox` tendra à avoir leurs bas à peu près alignés.

`\vtop` et `\vbox` sont souvent pratique quand vous voulez garder du texte ensemble sur une seule page. (Pour cela, peu importe normalement quelle commande vous utilisez.) Si votre usage de ces commandes empêche TEX de couper des pages d'une manière acceptable, TEX se plaindra qu'il a trouvé un "overfull" ou un "underfull vbox while `\output` is active".

La hauteur d'une vbox dépend des arguments de `\vtop` ou `\vbox`. Pour `\vbox`, TEX détermine la hauteur de la manière suivante :

- Si vous ne spécifiez que du ⟨*matériel en mode vertical*⟩, la vbox aura sa hauteur naturelle.
- Si vous spécifiez `to` ⟨*dimension*⟩, la hauteur de la vbox sera fixée à ⟨*dimension*⟩.
- Si vous spécifiez `spread` ⟨*dimension*⟩, la hauteur de la vbox sera sa hauteur naturelle plus ⟨*dimension*⟩, c'est-à-dire, que la hauteur de la vbox sera étirée verticalement de ⟨*dimension*⟩.

Pour `\vtop`, TEX construit la boîte en utilisant ses règles pour `\vbox` et ensuite répartit l'extension verticale entre la hauteur et la profondeur comme décrit ci-dessous.

Normalement, la largeur d'une vbox construite est la largeur de l'élément le plus largeur contenu[2]. Les règles de répartitions de l'extension verticale entre la hauteur et la profondeur sont plus compliquées :

- Pour `\vtop`, la hauteur est la hauteur de son premier élément, si cet élément est une boîte ou un filet. Autrement la hauteur est à zéro.

[2] Plus précisément, c'est la distance du point de référence au coté le plus à droite de la vbox construite. Par conséquent, si vous déplacez un des éléments vers la droite en utilisant `\moveright` ou `\moveleft` (avec une distance négative), la vbox construite pourra être plus large.

La profondeur est toute extension vertical restante quand la hauteur a été enlevée.

- Pour `\vbox`, la profondeur est la profondeur de son dernier élément, si cet élément est une boîte ou un filet. Autrement la profondeur est à zéro. La hauteur est toute extension vertical restante quand la profondeur a été enlevée[3].

La commande `\vfil` (p. 163) est pratique pour remplir une vbox avec de l'espace vide quand le matériel dans la boîte n'est pas aussi grand que l'extention verticale de la boîte.

Exemple :

```
\hbox{\hsize = 10pc \raggedright\parindent = 1em
\vtop{In this example, we see how to use vboxes to
produce the effect of double columns.  Each vbox
contains two paragraphs, typeset according to \TeX's
usual rules except that it's ragged right.\par
This isn't really the best way to get true double
columns because the columns}
\hskip 2pc
\vtop{\noindent
aren't balanced and we haven't done anything to choose
the column break automatically or even to fix up the
last line of the first column.\par
However, the technique of putting running text into a
vbox is very useful for placing that text where you
want it on the page.}}
```

produit :

In this example, we see how to use vboxes to produce the effect of double columns. Each vbox contains two paragraphs, typeset according to TEX's usual rules except that it's ragged right.

This isn't really the best way to get true double columns because the columns aren't balanced and we haven't done anything to choose the column break automatically or even to fix up the last line of the first column.

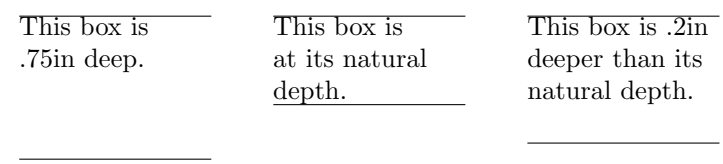
However, the technique of putting running text into a vbox is very useful for placing that text where you want it on the page.

[3] En fait, il ya une complication supplémentaire. Supposez qu'après que la profondeur ait été déterminée en utilisant les deux règles précédentes, la profondeur devienne plus grande que `\boxmaxdepth`. Alors la profondeur est réduite à `\boxmaxdepth` et la hauteur est ajustée en conséquence.

Exemple :

```
\hbox{\hsize = 1in \raggedright\parindent = 0pt
\vtop to .75in{\hrule This box is .75in deep. \vfil\hrule}
\qquad
\vtop{\hrule This box is at its natural depth. \vfil\hrule}
\qquad
\vtop spread .2in{\hrule This box is .2in deeper than
                  its natural depth.\vfil\hrule}}
```

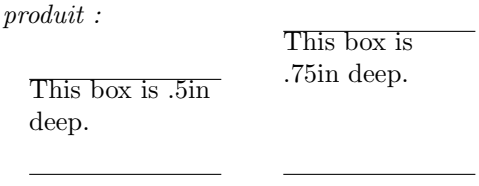
produit :



Exemple :

```
% See how \vbox lines up boxes at their bottoms
% instead of at their tops.
\hbox{\hsize = 1in \raggedright
\vbox to .5in{\hrule This box is .5in deep.\vfil\hrule}
\qquad
\vbox to .75in{\hrule This box is .75in deep.\vfil\hrule}}
```

produit :



`\boxmaxdepth` [⟨*dimension*⟩ paramètre]

Ce paramètre contient une dimension D. TEX ne construira pas une boîte dont la profondeur dépassera D. Si vous produisez une boîte dont la profondeur d doit dépasser D, TEX transfèrera la profondeur excédentaire dans la hauteur de la boîte, en abaissant en réalité le point de référence de la boîte de $d - D$. Si vous mettez `\boxmaxdepth` à zéro, TEX alignera une rangée de vbox de telle manière que leur frontière du bas seront toute sur la même ligne horizontale. Plain TEX met `\boxmaxdepth` à `\maxdimen` (p. 252), donc `\boxmaxdepth` n'affectera pas vos boîtes à moins que le changiez.

`\underbar` ⟨*argument*⟩

Cette commande met ⟨*argument*⟩ dans une hbox et la souligne sans faire attention à tout ce que dépasse sous la ligne de base de la boîte.

Exemple :

```
\underbar{Why not learn \TeX?}
```

produit :

Why not learn TEX?

`\everyhbox` [⟨*liste de token*⟩ paramètre]
`\everyvbox` [⟨*liste de token*⟩ paramètre]

Ces paramètres contiennent des liste de token que TEX développe au début de toutes hbox ou vbox qu'il construit. Tout élément résultant de l'expansion devient alors le début de la liste d'éléments pour la boîte. Par défaut, ces listes de token sont vide.

Remplir et récuperer le contenu de boîtes

`\setbox` ⟨*registre*⟩ = ⟨*boîte*⟩
`\box` ⟨*registre*⟩

Ces commandes remplissent et récupèrent respectivement le contenu du registre de boîte dont le numéro est ⟨*registre*⟩. Notez que vous remplissez un registre de boîte d'un manière un peu différente de celle utilisées pour les autres types de registres : vous utilisez `\setbox`*n* = rather than `\box`*n* =.

Récupérer le contenu d'un registre de boîte avec ces commandes a l'effet secondaire d'enlever ce qu'il contient, donc le registre de boîte devient vide. Si vous ne voulez pas que cela arrive, vous pouvez utiliser `\copy` (voir ci-dessous) pour récupérer le contenu. Vous devez utiliser `\box` de préférence à `\copy` quand vous n'avez plus besoin du contenu d'un registre de boîte après l'avoir utilisé, de manière à ne pas augmenter la mémoire de TEX en la remplissant avec des boîtes obsolètes.

Exemple :

```
\setbox0 = \hbox{mushroom}
\setbox1 = \vbox{\copy0\box0\box0}
\box1
```

produit :

mushroom
mushroom

`\copy` ⟨*registre*⟩

Cette commande produit une copy d'un registre de boîte ⟨*registre*⟩. Cette commande est utile quand vous voulez récupérer le contenu d'un registre

de boîte mais ne voulez pas en détruire le contenu. (Récupérer le contenu d'un registre avec `\box` rend le registre vide.)

Exemple :

```
\setbox0 = \hbox{good }
Have a \copy0 \box0 \box0 day!
```

produit :

Have a good good day!

`\unhbox` ⟨*registre*⟩
`\unvbox` ⟨*registre*⟩

Ces commandes produisent la liste contenue dans un registre de boîte ⟨*registre*⟩ et rend ce registre de boîte vide. `\unhbox` s'applique aux registres de boîte contenant des hbox et `\unvbox` s'applique aux registres de boîte contenant des vbox. Vous devez utiliser ces commandes en préférence à `\unhcopy` et `\unvcopy` (voir ci-dessous) quand vous n'avez pas besoin de ce qu'il y a dans le registre de boîte après l'avoir utilisé, de manière à ne pas augmenter la mémoire de TEX en la remplissant avec des boîtes obsolètes.

Exemple :

```
\setbox0=\hbox{The Mock Turtle sighed deeply, and
drew the back of one flapper across his eyes. }
\setbox1=\hbox{He tried to speak, but sobs choked
              his voice. }
\unhbox0 \unhbox1
% \box0 \box1 would set two hboxes side by side
% (and produce a badly overfull line).
\box1 % produces nothing
```

produit :

The Mock Turtle sighed deeply, and drew the back of one flapper across his eyes. He tried to speak, but sobs choked his voice.

`\unhcopy` ⟨*registre*⟩
`\unvcopy` ⟨*registre*⟩

Ces commandes produisent la liste contenue dans un registre de boîte ⟨*registre*⟩ et sans déranger le contenu des registres. `\unhbox` s'applique aux registres de boîte contenant des hbox et `\unvbox` s'applique aux registres de boîte contenant des vbox.

Exemple :

```
\setbox0=\hbox{The Mock Turtle sighed deeply, and
drew the back of one flapper across his eyes. }
\setbox1=\hbox{He tried to speak, but sobs choked his
          voice. }
\unhcopy0 \unhcopy1\par\noindent
% \box0 \box1 would set two hboxes side by side
% (and produce a badly overfull line).
\box1 % Produces an hbox (which can't be broken).
```

produit :

The Mock Turtle sighed deeply, and drew the back of one flapper across his eyes. He tried to speak, but sobs choked his voice.
He tried to speak, but sobs choked his voice.

Voir aussi : `\wd`, `\dp`, `\ht` (p. 173).

Déplacer des boîtes

`\moveleft` ⟨*dimension*⟩ ⟨*boîte*⟩
`\moveright` ⟨*dimension*⟩ ⟨*boîte*⟩

Ces commandes déplacent ⟨*boîte*⟩ à gauche ou à droite de ⟨*dimension*⟩ (qui peut être négatif). Vous ne pouvez appliquer `\moveleft` et `\moveright` qu'à des boîtes qui sont dans une liste verticale.

Exemple :

```
\vbox{\vbox{Phoebe}\vbox{walked}%
\moveleft 20pt\vbox{a}\moveright 20pt\vbox{crooked}%
\vbox{mile.}}
```

produit :

Phoebe
walked
a
crooked
mile.

`\lower` ⟨*dimension*⟩ ⟨*boîte*⟩
`\raise` ⟨*dimension*⟩ ⟨*boîte*⟩

Ces commandes déplacent ⟨*boîte*⟩ en haut ou en bas de ⟨*dimension*⟩ (qui peut être négatif). Vous ne pouvez appliquer `\moveleft` et `\moveright` qu'à des boîtes qui sont dans une liste horizontale.

Exemple :

```
Are you feeling \lower 6pt \hbox{depressed} about the
                \raise 6pt \hbox{bump} on your nose?
```

produit :

Are you feeling $_{\text{depressed}}$ about the $^{\text{bump}}$ on your nose?

Dimensions des registres de boîtes

`\ht` ⟨*registre*⟩ [⟨*dimension*⟩ paramètre]
`\dp` ⟨*registre*⟩ [⟨*dimension*⟩ paramètre]
`\wd` ⟨*registre*⟩ [⟨*dimension*⟩ paramètre]

Ces paramètres font référence à la hauteur, la profondeur et la largeur respectivement du registre de boîte ⟨*registre*⟩. Vous pouvez les utiliser pour connaître les dimensions d'une boîte. Vous pouvez aussi changer les dimensions d'une boîte, mais ce n'est pas une mince affaire ; si vous voulez être aventureux vous pouvez apprendre tout sur cela dans les pages 388–389 de *The TEXbook* et 999–999 de la traduction française.

Exemple :

```
\setbox0 = \vtop{\hbox{a}\hbox{beige}\hbox{bunny}}%
The box has width \the\wd0, height \the\ht0,
and depth \the\dp0.
```

produit :

The box has width 27.2223pt, height 4.30554pt, and depth 25.94444pt.

Struts, fantomes et boîtes vides

`\strut`

Cette commande produit une boîte dont la largeur est à zéro et dont la hauteur (`8.5pt`) et la profondeur (`3.5pt`) sont celle d'une ligne de caractère plus ou moins en `cmr10`, la police par défaut de plain TEX. Son utilisation principale est de forcer des lignes à avoir la même hauteur quand vous avez débranché le ressort inter-ligne de TEX avec `\offinterlineskip` ou une commande similaire, par exemple, quand vous construisez un alignement. Si la hauteur naturelle d'une ligne est trop petite, vous pouvez la ramener au standard en incluant un `\strut` dans la ligne. Le strut forcera la hauteur et la profondeur de la ligne à être plus grande, mais il n'imprime rien ni ne consomme aucun espace horizontal.

Si vous mettez des caractères d'une police qui est plus grande ou plus petite que `cmr10`, vous devez redéfinir `\strut` pour ce contexte.

Exemple :

```
\noindent % So we're in horizontal mode.
\offinterlineskip % So we get the inherent spacing.
% The periods in this vbox are not vertically equidistant.
\vtop{\hbox{.}\hbox{.(}\hbox{.x}
   \hbox{.\vrule height 4pt depth 0pt}}\qquad
% The periods in this vbox are vertically equidistant
% because of the struts.
\vtop{\hbox{.\strut}\hbox{.(\strut}\hbox{.x\strut}
   \hbox{.\vrule height 4pt depth 0pt\strut}}
```

produit :

`\mathstrut`

Cette commande produit une formule fantôme dont la largeur est à zéro et dont la largeur et la hauteur sont les mêmes que celle d'une parenthèse gauche. `\mathstrut` est en fait défini comme '`\vphantom(`'. Son utilisation principale est d'obtenir des radicaux, soulignés et surlignés alignés avec d'autres radicaux, soulignés et surlignés d'une formule. C'est un peu comme `\strut` (p. 173), sauf qu'il s'ajuste lui-même aux différents styles qui peuvent apparaître dans une formule mathématique.

Exemple :

```
$$\displaylines{
\overline{a_1a_2} \land \overline{b_1b_2}
\quad{\rm versus}\quad \overline{a_1a_2\mathstrut}
  \land \overline{b_1b_2\mathstrut}\cr
\sqrt{\epsilon} + \sqrt{\xi} \quad{\rm versus}\quad
\sqrt{\epsilon\mathstrut} + \sqrt{\xi\mathstrut}\cr}$$
```

produit :

$$\overline{a_1a_2} \land \overline{b_1b_2} \quad \text{versus} \quad \overline{a_1a_2} \land \overline{b_1b_2}$$
$$\sqrt{\epsilon} + \sqrt{\xi} \quad \text{versus} \quad \sqrt{\epsilon} + \sqrt{\xi}$$

`\phantom` ⟨*argument*⟩

Cette commande produit une boîte vide ayant la même taille et placement que ⟨*argument*⟩ aurait s'il était composé. Une utilisation de `\phantom` est de réserver de l'espace pour un symbole qui pour certaine raison demande à être dessiné à la main.

Exemple :

```
$1\phantom{9}2$
```

produit :
1 2

`\hphantom` ⟨*argument*⟩
`\vphantom` ⟨*argument*⟩
Ces commandes produisent des boîtes fantôme qui n'imprime rien :

- `\hphantom` produit une boîte avec la même largeur que ⟨*argument*⟩ mais de hauteur et profondeur zéro.
- `\vphantom` produit une boîte avec les mêmes hauteur et profondeur que ⟨*argument*⟩ mais de largeur zéro.

Leur usage principal est de forcer une sous-formule à avoir un certaine dimension horizontale ou verticale minimum.

Exemple :
```
$$\left[\vphantom{u\over v}t\right] \star
      \left[{u\over v}\right]\quad
   \{\hphantom{xx}\}$$
```
produit :

$$\left[t\right] \star \left[\frac{u}{v}\right] \quad \{ \quad \}$$

`\smash` ⟨*argument*⟩
Cette commande compose ⟨*argument*⟩, mais force la hauteur et la profondeur de sa boîte contenante à zéro. Vous pouvez utiliser `\smash` et `\vphantom` en combinaison pour donner à une sous-formule toutes hauteurs et profondeurs que vous désirez.

Exemple :
```
$${\smash{r_m \brace r_n}\vphantom{r}} \Longrightarrow r$$
```
produit :

$$\left\{\begin{matrix} r_m \\ r_n \end{matrix}\right\} \Longrightarrow r$$

`\null`
Cette commande produit une hbox vide.

Exemple :
```
\setbox0 = \null
The null box \null has width \the\wd0, height \the\ht0,
and depth \the\dp0.
```
produit :
The null box has width 0.0pt, height 0.0pt, and depth 0.0pt.

Paramètres faisant partie des boîtes mal formées

`\overfullrule` [⟨*dimension*⟩ paramètre]
Ce paramètre spécifie la largeur du filet que TEX positionne pour un "overfull hbox". Plain TEX le met à `5pt`.

`\hbadness` [⟨*nombre*⟩ paramètre]
`\vbadness` [⟨*nombre*⟩ paramètre]

Ces paramètres spécifient les seuils de badness horizontale et verticale pour rapporter des "underfull" ou "overfull boxes". `\hbadness` s'applique aux hbox et `\vbadness` s'applique aux vbox. Si la médiocrité d'une boîte construite dépasse le seuil, TEX reportera une erreur. Si vous augmentez les seuils (les valeurs par défaut de plain TEX sont de 1000), TEX sera moins amené à se plaindre. Notez que les valeurs de `\hbadness` et `\vbadness` n'ont pas d'effet sur l'apparence de votre document composé ; ils n'affectent que le message d'erreur que vous obtenez. Voir la page 302 de *The TEXbook* et 999 de la traduction française pour une description précise sur comment TEX décide quand se plaindre d'"overfull" ou d'"underfull box".

Exemple :

```
\hbadness = 10000 % Suppress any hbadness complaints.
\hbox to 2in{a b}\par
\hbadness = 500 % Report hbadness exceeding 500.
\hbox to 2in{a\hskip 0pt plus .5in b}
```

produit dans la log :

```
Underfull \hbox (badness 5091) detected at line 4
\tenrm a b

\hbox(6.94444+0.0)x144.54, glue set 3.70787
.\tenrm a
.\glue 0.0 plus 36.135
.\tenrm b
```

`\badness`

Cette commande donne la valeur numérique de la médiocrité de la boîte (aussi bien horizontale que verticale) que TEX a le plus récemment produit. Si la boîte était "overfull", `\badness` sera de 1000000 ; dans tous les autres cas il sera entre 0 et 10000.

`\hfuzz` [⟨*dimension*⟩ paramètre]
`\vfuzz` [⟨*dimension*⟩ paramètre]

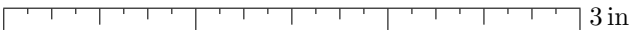
Ces paramètres spécifient les valeurs dont une boîte peut dépasser sa taille naturelle avant que TEX la considère comme étant overfull. `\hfuzz` s'applique aux hbox et `\vfuzz` s'applique aux vbox. Plain TEX mets ces deux paramètres à `0.1pt`.

Exemple :

```
\hfuzz = .5in
\hbox to 2in{This box is longer than two inches.}
% No error results
```

produit :

This box is longer than two inches.



Voir aussi : `\tolerance` (p. 128).

Retrouver le dernier élément d'une liste

`\lastkern`
`\lastskip`
`\lastpenalty`
`\lastbox`

Ces séquences de contrôle donnent la valeur du dernier élément de la liste courante. Ce ne sont pas de vraies commandes parce qu'elles ne peuvent apparaître que comme partie d'un argument. Si le dernier élément de la liste n'est pas du type indiqué, elles donnent une valeur zéro (ou une boîte vide, dans le cas de `\lastbox`). Par exemple, si le dernier élément de la liste courante est un crénage, `\lastkern` donne la dimension de ce crénage ; si ce n'est pas un crénage, il donne une dimension de 0.

Utiliser `\lastbox` a l'effet supplémentaire d'enlever la dernière boîte de la liste. Si vous voulez que le `\lastbox` original reste dans la liste, vous devez en ajouter une copie dans la liste. `\lastbox` n'est pas permise dans une liste mathématique ou dans la liste verticale principale.

Ces séquences de contrôle sont plus utiles après des appels de macro qui peuvent avoir inséré des entités du genre indiqué.

Exemple :

```
\def\a{two\kern 15pt}
one \a\a\hskip 2\lastkern three\par
% Get three times as much space before ‘three’.
\def\a{\hbox{two}}
one \a
\setbox0 = \lastbox % Removes ‘two’.
three \box0.
```

produit :

one two two three
one three two.

`\unkern`
`\unskip`
`\unpenalty`

Si le dernier élément de la liste courante est de type crénage, ressort ou pénalité respectivement, ces commandes l'enlèvent de cette liste. Si l'élément n'est pas du bon type, ces commandes n'ont pas d'effet. Comme

`\lastbox`, vous ne pouvez pas les appliquer à des listes en mode mathématique ou à la liste verticale principale. Ces commandes sont plus pratique après un appel de macro connu pour avoir inséré un élément spécifique que vous ne voulez pas ici. TEX ne procure pas de commande `\unbox` parce que `\lastbox` produit à peu près le même effet.

filets et règlures

`\hrule`
`\hrule height` ⟨*dimension*⟩ `width` ⟨*dimension*⟩ `depth` ⟨*dimension*⟩
`\vrule`
`\vrule width` ⟨*dimension*⟩ `height` ⟨*dimension*⟩ `depth` ⟨*dimension*⟩
La commande `\hrule` produit un filet horizontal ; La commande `\vrule` produit un filet vertical. Vous pouvez spécifier tout ou une partie des largeur, hauteur et profondeur du filet—TEX apporte des valeurs par défaut pour celles que vous omettez. Vous pouvez donner les dimensions du filet dans n'importe quel ordre ; les formes listées ci-dessus vous montrent simplement deux combinaisons possibles. Vous pouvez même donner une dimension d'un type donné plus d'une fois—si vous le faite, la dernière sera celle qui sera prise en compte.

Si vous ne spécifiez pas la largeur d'un filet horizontal, le filet est étendu horizontalement jusqu'aux limites de la boîte ou de l'alignement le plus interne qui contient le filet. Si vous ne spécifiez pas la hauteur d'un filet horizontal, elle sera par défaut de `0.4pt` ; si vous ne spécifiez pas la profondeur d'un filet horizontal, elle sera par défaut de `0pt`.

Si vous ne spécifiez pas la largeur d'un filet vertical, elle sera par défaut de `0.4pt`. Si vous ne spécifiez pas la hauteur d'un filet vertical, le filet est étendu jusqu'aux limites de la boîte ou de l'alignement le plus interne qui contient le filet.

TEX traite un filet horizontal comme un élément naturellement vertical et un filet vertical comme un élément naturellement horizontal. Donc un filet horizontal n'est légal qu'en mode vertical, Tandis qu'un filet vertical n'est légal qu'en mode horizontal. Si cela semble surprenant, visualisez le—un filet horizontal va de gauche à droite et sépare des éléments verticaux dans une séquence, tandis qu'un filet vertical va de haut en bas et sépare des éléments horizontaux dans une séquence.

Exemple :

```
\hrule\smallskip
\hrule width 2in \smallskip
\hrule width 3in height 2pt \smallskip
\hrule width 3in depth 2pt
```

produit :

Exemple :

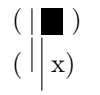
```
% Here you can see how the baseline relates to the
% height and depth of an \hrule.
\leftline{
   \vbox{\hrule width .6in height 5pt depth 0pt}
   \vbox{\hrule width .6in height 0pt depth 8pt}
   \vbox{\hrule width .6in height 5pt depth 8pt}
   \vbox{\hbox{ baseline}\kern 3pt \hrule width .6in}
}
```

produit :

Exemple :

```
\hbox{( {\vrule} {\vrule width 8pt} )}
\hbox {(( {\vrule height 13pt depth 0pt}
   {\vrule height 13pt depth 7pt} x)}
% the parentheses define the height and depth of each of the
% two preceding boxes; the 'x' sits on the baseline
```

produit :



☞ `\leaders` ⟨*boîte ou filet*⟩ ⟨*commande de saut*⟩
`\cleaders` ⟨*boîte ou filet*⟩ ⟨*commande de saut*⟩
`\xleaders` ⟨*boîte ou filet*⟩ ⟨*commande de saut*⟩

Ces commandes produisent des réglures, c'est-à-dire, qu'elles remplissent un espace horizontal ou vertical avec des copies d'un motif (voir "filets", p. 69). La ⟨*boîte*⟩ ou le ⟨*filet*⟩ spécifie une réglure, c'est-à-dire, une seule copie du motif, tandis que la ⟨*commande de saut*⟩ spécifie une fenêtre devant être remplie avec une rangée ou une colonne de réglures. Le motif est répété autant de fois qu'il peut rentrer dans la fenêtre. Si ⟨*commande de saut*⟩ est un saut horizontal, la fenêtre contient une rangée de réglures et TEX doit être en mode horizontal ; si ⟨*commande de saut*⟩ est un saut vertical, la fenêtre contient une colonne de réglures et TEX doit être en mode vertical.

Les commandes diffèrent dans leur façon d'arranger le motif répété dans l'espace et où elles mettent l'espace restant :

- Pour `\leaders`, TEX aligne une rangée de réglures avec la limite gauche de la boîte B la plus interne qui contient le résultat de la commande `\leaders`. Elle aligne une colonne de réglures avec le haut de B. Ces réglures qui tombent entièrement dans la fenêtre sont gardées. Tout espace en plus en haut et en bas de la fenêtre est laissé vide.
- Pour `\cleaders`, les réglures sont centrées dans la fenêtre.

- Pour `\xleaders` le motif est uniformément distribué à travers la fenêtre. Si l'espace en plus est l et que la réglure est répétée n fois, TEX met de l'espace de largeur ou hauteur $l/(n+1)$ entre des réglures adjacentes et aux deux bouts (gauche et droite ou haut et bas) des réglures.

Exemple :

```
\def\pattern{\hbox to 15pt{\hfil.\hfil}}
\line{Down the Rabbit-Hole {\leaders\pattern\hfil} 1}
\line{The Pool of Tears {\leaders\pattern\hfil} 9}
\line{A Caucus-Race and a Long Tale {\cleaders\pattern
      \hfil} 19}
\line{Pig and Pepper {\xleaders\pattern\hfil} 27}
```

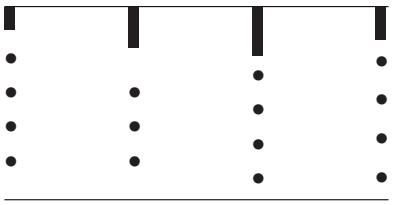
produit :

Down the Rabbit-Hole 1
The Pool of Tears 9
A Caucus-Race and a Long Tale 19
Pig and Pepper 27

Exemple :

```
\def\bulletfill{\vbox to 3ex{\vfil\hbox{$\bullet$}\vfil}}%
\def\mybox{\vbox to 1in}
\def\myrule{\hrule width 4pt}\hsize=2in
\hrule \line{%
   \mybox{\myrule depth 8pt \leaders\bulletfill\vfill}
   \hfil
   \mybox{\myrule depth 15pt \leaders\bulletfill\vfill}
   \hfil
   \mybox{\myrule depth 18pt \cleaders\bulletfill\vfill}
   \hfil
   \mybox{\myrule depth 12pt \xleaders\bulletfill\vfill}%
}\hrule
```

produit :



`\dotfill`
`\hrulefill`

Ces commandes remplissent respectivement l'espace horizontal englobé avec une rangée de points sur la ligne de base et avec une ligne horizontale sur la ligne de base. C'est habituellement une bonne idée de laisser une

espace entre `\dotfill` ou `\hrulefill` et le texte qui la précède ou la suit (voir l'exemple ci-dessous).

Exemple :

```
\hbox to 3in{Start {\dotfill} Finish}
\hbox to 3in{Swedish {\hrulefill} Finnish}
```

produit :

Start ..... Finish
Swedish _____ Finnish

`\leftarrowfill`
`\rightarrowfill`

Ces commandes remplissent l'espace horizontal englobé avec des flèches pointées vers la gauche ou vers la droite.

Exemple :

```
\hbox to 3in{\vrule \rightarrowfill \ 3 in
             \leftarrowfill\vrule}
```

produit :

|⟶ 3 in ⟵|

Alignements

Alignements tabulés

`\+` ⟨*texte*⟩ `&` ⟨*texte*⟩ `&` ⋯ `\cr`
`\tabalign`

Ces commandes débutent une simple ligne dans un alignement tabulé . La seule différence entre `\+` et `\tabalign` est que `\+` est une macro outer —vous ne pouvez pas l'utiliser quand TEX lit des tokens à haute vitesse (voir "outer", p. 84).

Si vous placez un '`&`' à une position à droite de toute tabulation existante dans un alignement tabulé, le '`&`' établit une nouvelle tabulation à cette position.

Exemple :

```
\cleartabs % Nullify any previous \settabs.
\+ {\bf if }$a[i] < a[i+1]$ &{\bf then}&\cr
\+&&$a[i] := a[i+1]$;\cr
\+&&{\it found }$:=$ {\bf true};\cr
\+&{\bf else}\cr
\+&&{\it found }$:=$ {\bf false};\cr
\+&{\bf end if};\cr
```

produit :

if $a[i] < a[i+1]$ **then**
$a[i] := a[i+1]$;
found := **true**;
else
found := **false**;
end if;

`\settabs` ⟨*nombre*⟩ `\columns`
`\settabs \+` ⟨*ligne d'exemple*⟩ `\cr`

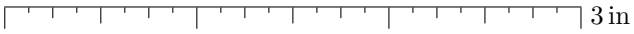
La première forme de cette commande définit un jeu de tabulations d'arrêt pour un alignement tabulé. Elle demande à TEX de mettre les tabulation d'arrêt de manière à divisez chaque ligne en ⟨*nombre*⟩ parties égales. TEX prend la longueur d'une ligne comme étant `\hsize`, normalement. Vous pouvez rendre l'alignement plus étroit en diminuant `\hsize`.

Exemple :

```
{\hsize = 3in \settabs 3 \columns
\+$1$&one&first\cr
\+$2$&two&second\cr
\+$3$&three&third\cr}
```

produit :

1 one first
2 two second
3 three third



La seconde forme de cette commande définit des tabulations d'arrêt en mettant les tabulations d'arrêt aux positions indiquées par les '`&` dans la ligne d'exemple. La ligne d'exemple elle-même ne doit pas apparaître dans la sortie. Quand vous utilisez cette forme vous devrez normalement mettre du matériel dans la ligne d'exemple qui sera un peu plus large que le plus large matériel correspondant dans l'alignement, de manière à produire de l'espace entre les colonnes. C'est ce que nous avons fait dans l'exemple ci-dessous. Le matériel suivant la dernière tabulation d'arrêt est inutile, puisqu'il n'a pas besoin de positionner quoi que ce soit à l'endroit où le `\cr` apparaît.

The tab settings established by `\settabs` remain in effect until you issue a new `\settabs` command or end a group containing the `\settabs` command. This is true for both forms of the command.

Exemple :

```
% The first line establishes the template.
\settabs \+$1$\qquad & three\quad & seventh\cr
\+$1$&one&first\cr
\+$2$&two&second\cr
\+$3$&three&third\cr
```

produit :

1 one first
2 two second
3 three third

`\cleartabs`
Cette commande efface toutes les tabulations à droite de la colonne courante. Son utilisation principale est dans des applications telle que la composition de programme informatique dans lesquels les positions de tabulation change à chaque ligne.

Voir aussi : `\cr`, `\endline`, `\crcr` (p. 187).

Alignement généraux

`\halign {` ⟨*préambule*⟩ `\cr` ⟨*rang*⟩ `\cr` ... ⟨*rang*⟩ `\cr }`
`\halign to` ⟨*dimension*⟩`{` ⟨*préambule*⟩ `\cr` ⟨*rang*⟩ `\cr` ... ⟨*rang*⟩ `\cr }`
`\halign spread` ⟨*dimension*⟩`{` ⟨*préambule*⟩ `\cr` ⟨*rang*⟩ `\cr` ... ⟨*rang*⟩ `\cr }`
Cette commande produit un alignement horizontal constitué d'un suite de rangées, où chaque rangée à son tour contient une suite d'entrée de colonne. TEX ajuste les largeurs des entrées de colonne pour accommoder la plus large dans chaque colonne.

Un alignement horizontal n'apparaît que quand TEX est dans un mode vertical. Nous vous recommandons d'apprendre les alignements en général (p. 45) avant d'essayer d'utiliser la commande `\valign`.

Un alignement consiste en un préambule suivi par le texte à aligner. Le préambule, qui décrit la forme des rangées qui le suivent, consiste en une suite de patrons de colonne, séparés par '`&`' et terminé par `\cr`. Chaque rangée consiste en une suite d'entrée de colonne, séparées aussi par '`&`' et terminées par `\cr`. Dans un patron, '`#`' indique où TEX doit insérer le texte correspondant à l'entrée de colonne. Par contre, `\settabs` utilise un patron fixe implicite à '`#`', c'est-à-dire, qu'il insère simplement le texte tel quel.

TEX compose chaque entrée de colonne en mode horizontal restreint, c'est-à-dire, comme le contenu d'une hbox, et englobe implicitement l'entrée dans un groupe.

La forme `to` de cette commande demande à TEX d'étendre la largeur de l'alignement à ⟨*dimension*⟩, en ajustant l'espace entre les colonnes. La forme `spread` de cette commande demande à TEX d'agrandir l'alignement de ⟨*dimension*⟩ par rapport à sa largeur naturelle. Ces formes sont comme les formes correspondantes de `\hbox` (p. 166).

Voir `\tabskip` (p. 190) pour un exemple d'utilisation de la forme `to`.

Exemple :

```
\tabskip = 1em \halign{%
   \hfil\it#\hfil&\hfil#\hfil&#&\hfil\$#\cr
   United States&Washington&dollar&1.00\cr
   France&Paris&franc&0.174\cr
   Israel&Jerusalem&shekel&0.507\cr
   Japan&Tokyo&yen&0.0829\cr}
```

produit :

United States	Washington	dollar	$1.00
France	Paris	franc	$0.174
Israel	Jerusalem	shekel	$0.507
Japan	Tokyo	yen	$0.0829

`\valign {` ⟨*préambule*⟩`\cr` ⟨*colonne*⟩`\cr` ... ⟨*colonne*⟩`\cr }`
`\valign to` ⟨*dimension*⟩`{` ⟨*préambule*⟩`\cr` ⟨*colonne*⟩`\cr` ... ⟨*colonne*⟩`\cr }`
`\valign spread` ⟨*dimension*⟩`{` ⟨*préambule*⟩`\cr` ⟨*col.*⟩`\cr` ... ⟨*col.*⟩`\cr }`

Cette commande produit un alignement vertical constitué d'un suite de colonnes, où chaque colonne à son tour contient une suite d'entrée de rangée. TEX ajuste les hauteurs des entrées de rangée pour accommoder la plus longue dans chaque rangée.

Un alignement vertical n'apparaît que quand TEX est dans un mode horizontal. Puisque les alignements verticaux sont (a) conceptuellement assez difficile et (b) peu souvent utilisé, nous vous recommandons d'apprendre les alignements en général (p. 45) et la commande `\halign` (voir ci-dessus) avant d'essayer d'utiliser la commande `\valign`.

Un alignement consiste en un préambule suivi par le texte à aligner. Le préambule, qui décrit la forme des colonnes qui le suivent, consiste en une suite de patrons de rangée, séparés par '`&`' et terminé par `\cr`. Chaque colonne consiste en une suite d'entrée de rangée, séparées aussi par '`&`' et terminées par `\cr`. Dans un patron, '`#`' indique où TEX doit insérer le texte correspondant à l'entrée de rangée.

TEX compose chaque entrée de rangée en mode vertical interne, c'est-à-dire, comme le contenu d'une vbox, et englobe implicitement l'entrée dans un groupe. Il donne toujours à la vbox une profondeur à zéro. Tout texte ou autre matériel en mode horizontal dans une entrée de rangée met alors TEX en mode horizontal ordinaire. (c'est juste une application des règles générales du fonctionnement de TEX en mode vertical interne.) Les paramètres de paragraphage habituels s'appliquent dans ce cas : l'entrée de rangée a une indentation initial de `\parindent` (p. 119) et les ressorts `\leftskip` et `\rightskip` (p. 121) s'appliquent à ses lignes.

Notez en particulier qu'une entrée de rangée contenant du texte a une largeur de `\hsize` (p. 120). A moins que vous mettiez `\hsize` à la largeur de rangée que vous voulez, vous rencontrerez vraisemblablement des "overfull hbox" ou trouverez que la première colonne occupe la largeur de la page entière, ou les deux.

Normalement, vous devrez inclure un strut dans chaque patron pour que les rangées n'apparaissent pas comme un résultat des hauteurs variables des entrées de l'alignement. Vous pouvez produire un strut avec la commande `\strut`.

La forme `to` de cette commande demande à TEX d'étendre l'extension verticale de l'alignement à ⟨*dimension*⟩, en ajustant l'espace entre les rangées. La forme `spread` de cette commande demande à TEX d'agrandir l'alignement de ⟨*dimension*⟩ par rapport à sa hauteur naturelle. Ces formes sont comme les formes correspondantes de `\vbox` (p. 167).

Exemple :

```
{\hsize=1in \parindent=0pt
\valign{#\strut&#\strut&#\strut&#\strut\cr
   bernaise&curry&hoisin&hollandaise\cr
   ketchup&marinara&mayonnaise&mustard\cr
   rarebit&tartar\cr}}
```

produit :

bernaise	ketchup	rarebit
curry	marinara	tartar
hoisin	mayonnaise	
hollandaise	mustard	

Exemple :

```
% same thing but without struts (shows why you need them)
{\hsize=1in \parindent=0pt
\valign{#&#&#&#\cr
   bernaise&curry&hoisin&hollandaise\cr
   ketchup&marinara&mayonnaise&mustard\cr
   rarebit&tartar\cr}}
```

produit :

bernaise	ketchup	rarebit
curry	marinara	tartar
hoisin	mayonnaise	
hollandaise	mustard	

`\ialign`

Cette commande réagit comme `\halign`, sauf qu'elle met d'abord le ressort `\tabskip` à zéro et met `\everycr` à vide.

`\cr`

Cette commande termine le préambule d'un alignement horizontal ou vertical, une rangée d'un alignement horizontal ou tabulé, ou une colonne d'un alignement vertical. Vous pouvez demander à TEX de faire certaines actions à chaque fois qu'il rencontre un `\cr` en chargeant la valeur du paramètre `\everycr` (p. 192).

`\endline`

Cette commande est un synonyme de la commande `\cr`. Elle est utile quand vous avez redéfini `\cr` mais devez encore accéder à sa définition originale.

`\crcr`

Cette commande réagit comme `\cr`, sauf que TEX l'ignore si elle arrive immédiatement après un `\cr` ou un `\noalign`. Sa principale application est comme mesure de sécurité pour éviter un message d'erreur "misleading" causé par une macro qui attend un argument se terminant par `\cr`. Si vous mettez `\crcr` après le '`#`*n*' qui dénote un tel argument dans la définition de macro, la macro marchera proprement, que l'argument se termine par `\cr` ou non.

`\omit`

Cette commande demande à TEX d'ignorer un patron dans un alignement horizontal ou vertical pendant le traitement d'une entrée de colonne ou de rangée particulière respectivement. `\omit` doit apparaître comme le premier élément dans une entrée de colonne ou de rangée ; en effet, il remplace le patron du préambule avec le simple patron '`#`'.

Exemple :

```
\tabskip = 2em\halign{%
   \hfil\it#\hfil&\hfil#\hfil&#&\hfil\$#\cr
   United States&Washington&dollar&1.00\cr
   \omit \dotfill France\dotfill&Paris&franc&0.174\cr
   Israel&Jerusalem&shekel&0.507\cr
   Japan&Tokyo&yen&0.0829\cr}
```

produit :

United States	Washington	dollar	$1.00
...France...	Paris	franc	$0.174
Israel	Jerusalem	shekel	$0.507
Japan	Tokyo	yen	$0.0829

Exemple :

```
{\hsize=1.2in \parindent=0pt
\valign{(#)\strut&(#)\strut&(#)\strut&(#)\strut\cr
   bernaise&curry&hoisin&hollandaise\cr
   ketchup&\omit\strut{\bf MARINARA!}&mayonnaise&mustard\cr

   rarebit&tartar\cr}}
```

produit :

(bernaise)	(ketchup)	(rarebit)
(curry)	**MARINARA!**	(tartar)
(hoisin)	(mayonnaise)	
(hollandaise)	(mustard)	

`\span`

La signification de cette commande dépend qu'elle apparaît soit dans un préambule soit dans une entrée d'alignement.

- Normalement, TEX ne développe pas de tokens dans le préambule quand il le lit. Mettre `\span` au début d'un token dans le préambule rend ce token développable immédiatement selon les règles usuelles de développement de macro de TEX.
- Mettre `\span` à la place de '`&`' entre deux entrées de colonne ou de rangée rend ces colonnes ou rangées combinées. Pour un alignement horizontal, la largeur de la colonne combinée est la somme des largeurs des colonnes incluses. Pour un alignement vertical, la hauteur de la rangée combinée est la somme des hauteurs des rangées incluses. Le patron des colonnes ou rangées combinées forme un seul groupe, donc des commandes de chargement de police précédant un `\span` affectent tout jusqu'au prochain '`&`'.

`\span` est rarement utile d'elle-même en dehors d'un patron, mais procure le mécanisme de base pour définir `\multispan`.

`\multispan` ⟨*nombre*⟩

Cette commande demande à TEX que les ⟨*nombre*⟩ colonnes suivantes d'une rangée d'un alignement horizontal, ou que les ⟨*nombre*⟩ rangées dans une colonne d'un alignement vertical, doivent être combinées en une seule colonne ou rangée (comme avec `\span`) et que leur patron doit être omis (comme avec `\omit`).

Exemple :

```
\tabskip = 13pt\halign{%
   \hfil\it#\hfil&\hfil#\hfil&#&\hfil\$#\cr
   United States&Washington&dollar&1.00\cr
   France&Paris&franc&0.174\cr
   Israel&Jerusalem &
      \multispan 2 \hfil\it(no information)\hfil \cr
   Japan&Tokyo&yen&0.0829\cr}
```

produit :

United States	Washington	dollar	$1.00
France	Paris	franc	$0.174
Israel	Jerusalem	*(no information)*	
Japan	Tokyo	yen	$0.0829

Exemple :

```
{\hsize=1.2in \parindent=0pt
\valign{(#)\strut&(#)\strut&(#)\strut&(#)\strut\cr
   bernaise&curry&hoisin&hollandaise\cr
   \multispan 3$$\left\{{{\rm ketchup}\atop{\rm marinara}}
      \right\}$$&mustard\cr
   rarebit&tartar\cr}}
```

produit :

(bernaise)
(curry)
(hoisin)
$\left\{\begin{array}{l}\text{ketchup}\\ \text{marinara}\end{array}\right\}$
(rarebit)
(tartar)

(hollandaise) (mustard)

`\noalign {` ⟨*matériel en mode vertical*⟩ `}`
`\noalign {` ⟨*matériel en mode horizontal*⟩ `}`

Cette commande insère ⟨*matériel en mode vertical*⟩ après la rangée courante d'un alignement horizontal ou ⟨*matériel en mode horizontal*⟩ après la colonne courante d'un alignement vertical. Le matériel peut être du texte, un ressort, un filet ou tout autre chose.

L'utilisation la plus commune de `\noalign` est de mettre de l'espace supplémentaire après une rangée ou une colonne. Si vous voulez mettre de l'espace supplémentaire après *toutes* les rangées d'un alignement horizontal, utilisez `\openup` (p. 141).

Exemple :

```
\halign{%
   \hfil\it#\hfil\tabskip=2em&\hfil#\hfil&##&
      \hfil\$#\tabskip=0em\cr
   % The \tabskip changes prevent the rule below
   % from sticking out.
   United States&Washington&dollar&1.00\cr
   France&Paris&franc&0.174\cr
   \noalign{\smallskip\hrule\smallskip}
   Israel&Jerusalem&shekel&0.507\cr
   Japan&Tokyo&yen&0.0829\cr}
```

produit :

United States	Washington	dollar	$1.00
France	Paris	franc	$0.174
Israel	Jerusalem	shekel	$0.507
Japan	Tokyo	yen	$0.0829

Exemple :

```
{\hsize=1in \parindent=0pt
\valign{#\strut&#\strut&#\strut&#\strut\cr
   \noalign{\vrule width 2pt\quad}
   bernaise&curry&hoisin&hollandaise\cr
   \noalign{\vrule width 2pt\quad}
   ketchup&marinara&mayonnaise&mustard\cr
   \noalign{\vrule width 2pt\quad}
   rarebit&tartar\cr
   \noalign{\vrule width 2pt\quad}}}
```

produit :

bernaise	ketchup	rarebit	
curry	marinara	tartar	
hoisin	mayonnaise		
hollandaise	mustard		

`\tabskip` [⟨*ressort*⟩ paramètre]

Ce paramètre spécifie le montant de ressort horizontal ou vertical que TEX met entre les colonnes d'un alignement horizontal ou entre les rangées d'un alignement vertical. TEX met aussi le ressort `\tabskip` à gauche de la première colonne et à droite de la dernière colonne d'un alignement horizontal et avant la première rangée et après la dernière rangée d'un alignement vertical. Vous pouvez changer `\tabskip` dans un patron—le changement affectera le ressort associé avec tous les `&` suivant aussi bien que le ressort après la dernière rangée ou colonne.

Exemple :

```
\halign to 3.5in{%
   \hfil\it#\tabskip = 2em plus 8pt
      \hfil&\hfil#\hfil&#\tabskip = 1em
      &\hfil\$#\tabskip = 0em\cr
   United States&Washington&dollar&1.00\cr
   France&Paris&franc&0.174\cr
   Israel&Jerusalem&shekel&0.507\cr
   Japan&Tokyo&yen&0.0829\cr}
```

produit :

United States	Washington	dollar	$1.00
France	Paris	franc	$0.174
Israel	Jerusalem	shekel	$0.507
Japan	Tokyo	yen	$0.0829

Exemple :

```
{\hsize = 1in \parindent=0pt \tabskip=5pt
\valign{#\strut&#\strut\tabskip = 3pt
   &#\strut&#\strut\cr
   bernaise&curry&hoisin&hollandaise\cr
   ketchup&marinara&mayonnaise&mustard\cr
   rarebit&tartar\cr}}
```

produit :

bernaise	ketchup	rarebit
curry	marinara	tartar
hoisin	mayonnaise	
hollandaise	mustard	

`\hidewidth`

Cette commande demande à TEX d'ignorer la largeur de la l'entrée de la colonne suivante dans un alignement horizontal. c'est pratique quand vous avez une entrée qui est plus longue que la plupart des autres dans la même colonne et que vous préférez avoir cette entrée sortant de la colonne plutôt que rendre toutes les entrées de la colonne plus large. Si le `\hidewidth` est à gauche de l'entrée, l'entrée débordera à gauche ; si le `\hidewidth` est à droite de l'entrée, l'entrée débordera à droite.

Exemple :

```
\tabskip = 25pt\halign{%
   \hfil\it#\hfil&\hfil#\hfil&#&\hfil\$#\cr
   United States&\hidewidth Washington&
      dollar&1.00\cr
   France&Paris&franc&0.174\cr
   Israel&Jerusalem&shekel&0.507\cr
   Japan&Tokyo&yen&0.0829\cr}
```

produit :

United States	Washington	dollar	$1.00
France	Paris	franc	$0.174
Israel	Jerusalem	shekel	$0.507
Japan	Tokyo	yen	$0.0829

`\everycr` [⟨*liste de token*⟩ paramètre]

TEX développe ⟨*token list*⟩ à chaque fois qu'il exécute un `\cr`—à la fin de chaque préambule, à la fin de chaque rangée d'un alignement horizontal et à la fin de chaque colonne d'un alignement vertical. La commande `\everycr` est développée juste après le `\cr`. Ainsi vous pouvez demander à TEX d'exécuter certaines commandes à la fin d'un préambule, d'un rangée ou d'une colonne en assignant une liste de commandes à `\everycr`.

Les tokens `\everycr` ne doivent pas inclure d'autre commandes que `\noalign`. Ceci parce que les tokens `\everycr` réapparaîtrons après le dernier `\cr` de l'alignement. Une commande autre qu'un `\noalign` fera penser à TEX qu'il commence une nouvelle rangée ou colonne. TEX se plaindra d'un `\cr` manquant, insèrera un `\cr`, insèrera les tokens `\everycr` à nouveau et répètera ces actions indéfiniment.

Exemple :

```
\everycr={\noalign{\smallskip\hrule\smallskip}}
\halign{#\tabskip = 11pt&\hfil#\hfil&\hfil#\hfil
      \tabskip = 0pt\cr
   $1$&one&first\cr
   $2$&two&second\cr
   $3$&three&third\cr}
```

produit :

1	one	first
2	two	second
3	three	third

8 Commandes pour composer des formules mathématiques

Cette section couvre les commandes pour construire des formules mathématiques. Pour une explication des conventions utilisées dans cette section, voir "Description des commandes" (p. 3).

Parties simples de formules

Lettres grecques

☞	α	\alpha	μ	\mu	σ	\sigma
	β	\beta	ν	\nu	ς	\varsigma
	χ	\chi	ω	\omega	Σ	\Sigma
	δ	\delta	Ω	\Omega	τ	\tau
	Δ	\Delta	ϕ	\phi	θ	\theta
	ϵ	\epsilon	φ	\varphi	ϑ	\vartheta
	ε	\varepsilon	Φ	\Phi	Θ	\Theta
	η	\eta	π	\pi	υ	\upsilon
	γ	\gamma	ϖ	\varpi	Υ	\Upsilon
	Γ	\Gamma	Π	\Pi	ξ	\xi
	ι	\iota	ψ	\psi	Ξ	\Xi
	κ	\kappa	Ψ	\Psi	ζ	\zeta
	λ	\lambda	ρ	\rho		
	Λ	\Lambda	ϱ	\varrho		

Ces commandes produisent les lettres grecques appropriées aux mathématiques. Vous ne pouvez les utiliser que dans une formule mathématique, ainsi si vous avez besoin d'une lettre grecque dans du texte ordinaire, vous devez l'englober entre des signes dollar (\$). TEX n'a pas

de commandes pour des lettres grecques qui ressemblent à leurs contreparties romaines, puisque vous pouvez les obtenir en employant ces contreparties romaines. Par exemple, vous pouvez obtenir un omicron minuscule dans une formule en écrivant la lettre 'o', c'est-à-dire, '`{\rm o}`' ou un beta majuscule ('B') en écrivant '`{\rm B}`'.

Ne confondez pas les lettres suivantes :

- `\upsilon` ('υ'), `{\rm v}` ('v'), and `\nu` ('ν').
- `\varsigma` ('ς') and `\zeta` ('ζ').

vous pouvez obtenir des lettres grecques capitales inclinées en utilisant la police mathématique italique (`\mit`).

TEX traite les lettres grecques comme des symboles ordinaires quand il calcule combien d'espace mettre autour d'eux.

Exemple :

```
If $\rho$ and $\theta$ are both positive, then $f(\theta)
-{\mit \Gamma}_{\theta} < f(\rho)-{\mit \Gamma}_{\rho}$.
```

produit :

If ρ and θ are both positive, then $f(\theta) - \Gamma_\theta < f(\rho) - \Gamma_\rho$.

Divers Symboles mathématiques ordinaires

☞	∞	`\infty`	$\exists$	`\exists`	∂	`\partial`
	$\Re$	`\Re`	$\forall$	`\forall`	$\surd$	`\surd`
	$\Im$	`\Im`	$\hbar$	`\hbar`	$\wp$	`\wp`
	$\angle$	`\angle`	ℓ	`\ell`	$\flat$	`\flat`
	$\triangle$	`\triangle`	$\aleph$	`\aleph`	$\sharp$	`\sharp`
	$\backslash$	`\backslash`	$\imath$	`\imath`	$\natural$	`\natural`
	$\vert$	`\vert`	$\jmath$	`\jmath`	$\clubsuit$	`\clubsuit`
	$\Vert$	`\\|`	∇	`\nabla`	$\diamondsuit$	`\diamondsuit`
	$\Vert$	`\Vert`	$\neg$	`\neg`	$\heartsuit$	`\heartsuit`
	$\emptyset$	`\emptyset`	$\neg$	`\lnot`	$\spadesuit$	`\spadesuit`
	$\bot$	`\bot`	$'$	`'` (apostrophe)		
	$\top$	`\top`	$\prime$	`\prime`		

Ces commandes produisent des symboles variés. On les appelle "symboles ordinaires" pour les distinguer des autres classes de symboles comme les relations. Vous ne pouvez utiliser un symbole ordinaire que dans une formule mathématique, donc si vous avez besoin d'un symbole ordinaire dans du texte ordinaire vous devez l'englober entre des signes dollar (`$`).

Les commandes `\imath` et `\jmath` sont utiles quand vous devez mettre un accent sur un 'i' ou un 'j'.

Une apostrophe (`'`) est un raccourci pour écrire un `\prime` en exposant. (La commande `\prime` de son côté, génère un gros prime laid.)

Les commandes `\|` et `\Vert` sont synonymes, comme le sont les commandes `\neg` et `\lnot`. La commande `\vert` produit le même résultat que '`|`'.

Les symboles produit par `\backslash`, `\vert` et `\Vert` sont des délimiteurs. ces symboles peuvent être produit en grandes taille en utilisant `\bigm` etc. (p. 219).

Exemple :

```
The Knave of $\heartsuit$s, he stole some tarts.
```

produit :

The Knave of ♡s, he stole some tarts.

Exemple :

```
If $\hat\imath < \hat\jmath$ then $i' \leq j^\prime$.
```

produit :

If $\hat{\imath} < \hat{\jmath}$ then $i' \leq j'$.

Exemple :

```
$${{x-a}\over{x+a}}\biggm\backslash{{y-b}\over{y+b}}$$
```

produit :

$$\frac{x-a}{x+a} \Big\backslash \frac{y-b}{y+b}$$

Opérations binaires

☞	$\vee$	`\vee`	$\cdot$	`\cdot`	$\triangleleft$	`\triangleleft`
	$\wedge$	`\wedge`	$\diamond$	`\diamond`	$\triangleright$	`\triangleright`
	$\amalg$	`\amalg`	$\bullet$	`\bullet`	$\bigtriangledown$	`\bigtriangledown`
	$\cap$	`\cap`	$\circ$	`\circ`	$\bigtriangleup$	`\bigtriangleup`
	$\cup$	`\cup`	$\bigcirc$	`\bigcirc`	$\ast$	`\ast`
	$\uplus$	`\uplus`	$\odot$	`\odot`	$\star$	`\star`
	$\sqcap$	`\sqcap`	$\ominus$	`\ominus`	$\times$	`\times`
	$\sqcup$	`\sqcup`	$\oplus$	`\oplus`	$\div$	`\div`
	$\dagger$	`\dagger`	$\oslash$	`\oslash`	$\setminus$	`\setminus`
	$\ddagger$	`\ddagger`	$\otimes$	`\otimes`	$\wr$	`\wr`
	$\wedge$	`\land`	$\pm$	`\pm`		
	$\vee$	`\lor`	$\mp$	`\mp`		

Ces commandes produisent les symboles pour différentes opérations binaires. Les opération binaire sont une des classes des symboles mathématiques de TEX. TEX mets différent montants d'espace autour de différentes classes de symboles mathématiques. Quand TEX a besoin de couper une ligne de texte dans une formule mathématique, il devra placer la coupure après une opération binaire—mais seulement si l'opération est à un niveau supérieur à la formule, c'est-à-dire, non incluse dans un groupe.

En plus de ces commandes, TEX traite également '`+`' et '`-`' en tant qu'opérations binaires. Il considère '`/`' comme un symbole ordinaire, en

dépit du fait que, mathématiquement, ce soit une opération binaire, parce qu'il semble mieux avec moins d'espace autour de lui.

Exemple :

```
$$z = x \div y \quad \hbox{if and only if} \quad
z \times y = x \;\hbox{and}\; y \neq 0$$
```

produit :

$$z = x \div y \quad \text{if and only if} \quad z \times y = x \text{ and } y \neq 0$$

`\*`

La commande `\*` indique un symbole de multiplication optionnel ($\times$), qui est une opération binaire. Ce symbole de multiplication agit comme une césure optionnelle quand il apparaît dans une formule dans un texte. Cela étant, TEX ne composera le symbole `\times` *que* si la formule doit être coupé en ce point. Il n'y a aucun point en utilisant `\*` dans des formule affichées car TEX ne coupe jamais de formules affichées de lui-même.

Exemple :

```
Let $c = a\*b$. In the case that $c=0$ or $c=1$, let
$\Delta$ be $(\hbox{the smallest $q$})\*(\hbox{the
largest $q$})$ in the set of approximate $\tau$-values.
```

produit :

Let $c = ab$. In the case that $c = 0$ or $c = 1$, let Δ be (the smallest q) $\times$ (the largest q) in the set of approximate τ-values.

Relations

$\asymp$ `\asymp`	$\gg$ `\gg`	$\bowtie$ `\bowtie`
$\cong$ `\cong`	$\ll$ `\ll`	$\propto$ `\propto`
$\dashv$ `\dashv`	$\models$ `\models`	$\approx$ `\approx`
$\vdash$ `\vdash`	$\neq$ `\ne`	$\sim$ `\sim`
$\perp$ `\perp`	$\neq$ `\neq`	$\simeq$ `\simeq`
$\mid$ `\mid`	$\notin$ `\notin`	$\frown$ `\frown`
$\parallel$ `\parallel`	$\in$ `\in`	$\smile$ `\smile`
$\doteq$ `\doteq`	$\ni$ `\ni`	$\subset$ `\subset`
$\equiv$ `\equiv`	$\ni$ `\owns`	$\subseteq$ `\subseteq`
$\geq$ `\ge`	$\prec$ `\prec`	$\supset$ `\supset`
$\geq$ `\geq`	$\preceq$ `\preceq`	$\supseteq$ `\supseteq`
$\leq$ `\le`	$\succ$ `\succ`	$\sqsubseteq$ `\sqsubseteq`
$\leq$ `\leq`	$\succeq$ `\succeq`	$\sqsupseteq$ `\sqsupseteq`

Ces commands produisent des symboles pour différentes relations. Les relations sont une des classes des symboles mathématiques. TEX mets différent montants d'espace autour de différentes classes de symboles mathématiques. Quand TEX a besoin de couper une ligne de texte dans

une formule mathématique, il devra placer la coupure après une relation—mais seulement si l'opération est à un niveau supérieur à la formule, c'est-à-dire, non incluse dans un groupe.

En plus des commandes listées ici, TEX traite '=' et les commandes "flèche" (p. 200) comme des relations.

Certaines relations ont plus d'une commande que vous pouvez utiliser pour les produire :

- '≥' (`\ge` et `\geq`).
- '≤' (`\le` et `\leq`).
- '≠' (`\ne`, `\neq` et `\not=`).
- '∋' (`\ni` et `\owns`).

Vous pouvez produire des relations négatives en les préfixant avec `\not`, comme cela :

$\not\asymp$ `\not\asymp`	$\not\leq$ `\not\leq`	$\not\simeq$ `\not\simeq`
$\not\cong$ `\not\cong`	$\not\prec$ `\not\prec`	$\not\subset$ `\not\subset`
$\not\equiv$ `\not\equiv`	$\not\preceq$ `\not\preceq`	$\not\subseteq$ `\not\subseteq`
$\not=$ `\not=`	$\not\succ$ `\not\succ`	$\not\supset$ `\not\supset`
$\not\geq$ `\not\ge`	$\not\succeq$ `\not\succeq`	$\not\supseteq$ `\not\supseteq`
$\not\geq$ `\not\geq`	$\not\approx$ `\not\approx`	$\not\sqsubseteq$ `\not\sqsubseteq`
$\not\leq$ `\not\le`	$\not\sim$ `\not\sim`	$\not\sqsupseteq$ `\not\sqsupseteq`

Exemple :

```
We can show that $AB \perp AC$, and that
$\triangle ABF \not\sim \triangle ACF$.
```

produit :

We can show that $AB \perp AC$, and that $\triangle ABF \not\sim \triangle ACF$.

Délimiteurs gauches et droits

{ `\lbrace`	[`\lbrack`	⌈ `\lceil`
{ `\{`	] `\rbrack`	⌉ `\rceil`
} `\rbrace`	⟨ `\langle`	⌊ `\lfloor`
} `\}`	⟩ `\rangle`	⌋ `\rfloor`

Ces commandes produisent des délimiteurs gauches et droits. Les mathématiciens utilisent les délimiteurs pour indiquer les frontières entre les parties d'une formule. Les délimiteurs gauches sont aussi appelés "ouvrants" et les délimiteurs droits sont aussi appelés "fermants". Ouvrants et Fermants sont deux des classes de symboles mathématiques de TEX. TEX mets différents montants d'espace autour des différentes classes de symboles mathématiques. Vous devriez vous attendre à ce que l'espacement que TEX mets autour des ouvrants et des fermants soit symétrique, mais en fait, il ne l'est pas.

Certains délimiteurs gauches et droits ont plus d'une commande utilisable pour les produire :

- '{' (`\lbrace` and `\{`)

- '}' (`\rbrace` and `\}`)
- '[' (`\lbrack` and '`[`')
- ']' (`\rbrack` and '`]`')

vous pouvez aussi utiliser les caractères crochets gauches et droits (sous toutes ses forme) en dehors du mode mathématique.

En plus de ces commandes, TEX traite '`(`' comme un délimiteur gauche et '`)`' comme un délimiteur droit.

Vous pouvez laisser TEX choisir la taille d'un délimiteur en utilisant `\left` et `\right` (p. 212). Alternativement, vous pouvez obtenir un délimiteur d'une taille spécifique en utilisant une des commandes `\big`*x* (voir `\big` et al., p. 219).

Exemple :

```
The set $\{\,x \mid x>0\,\}$ is empty.
```

produit :

The set $\{\,x \mid x>0\,\}$ is empty.

Flèches

☞	$\leftarrow$	`\leftarrow`	$\leftharpoondown$	`\leftharpoondown`
	$\leftarrow$	`\gets`	$\rightharpoondown$	`\rightharpoondown`
	$\Leftarrow$	`\Leftarrow`	$\leftharpoonup$	`\leftharpoonup`
	$\rightarrow$	`\rightarrow`	$\rightharpoonup$	`\rightharpoonup`
	$\rightarrow$	`\to`	$\rightleftharpoons$	`\rightleftharpoons`
	$\Rightarrow$	`\Rightarrow`	$\mapsto$	`\mapsto`
	$\leftrightarrow$	`\leftrightarrow`	$\longmapsto$	`\longmapsto`
	$\Leftrightarrow$	`\Leftrightarrow`	$\downarrow$	`\downarrow`
	$\longleftarrow$	`\longleftarrow`	$\Downarrow$	`\Downarrow`
	$\Longleftarrow$	`\Longleftarrow`	$\uparrow$	`\uparrow`
	$\longrightarrow$	`\longrightarrow`	$\Uparrow$	`\Uparrow`
	$\Longrightarrow$	`\Longrightarrow`	$\updownarrow$	`\updownarrow`
	$\longleftrightarrow$	`\longleftrightarrow`	$\Updownarrow$	`\Updownarrow`
	$\Longleftrightarrow$	`\Longleftrightarrow`	$\nearrow$	`\nearrow`
	$\iff$	`\iff`	$\searrow$	`\searrow`
	$\hookleftarrow$	`\hookleftarrow`	$\nwarrow$	`\nwarrow`
	$\hookrightarrow$	`\hookrightarrow`	$\swarrow$	`\swarrow`

Ces commandes procurent des flèches de différentes sortes. Elles sont classées parmi les relations (p. 198). Les flèches verticales de la liste sont aussi des délimiteurs, donc vous pouvez les rendre plus grandes en utilisant `\big` et al. (p. 219).

La commande `\iff` diffère de `\Longleftrightarrow` parce qu'elle produit un espace supplémentaire à gauche et à droite de la flèche.

Vous pouvez placer des symboles ou d'autres légendes au dessus des flèche gauches et droite avec `\buildrel` (p. 210).

Exemple :

```
$$f(x)\mapsto f(y) \iff x \mapsto y$$
```

produit :

$$f(x) \mapsto f(y) \iff x \mapsto y$$

Fonctions mathématiques nommées

☞	cos	`\cos`	sinh	`\sinh`	hom	`\hom`
	sin	`\sin`	tanh	`\tanh`	ker	`\ker`
	tan	`\tan`	det	`\det`	inf	`\inf`
	cot	`\cot`	dim	`\dim`	sup	`\sup`
	csc	`\csc`	exp	`\exp`	lim	`\lim`
	sec	`\sec`	ln	`\ln`	lim inf	`\liminf`
	arccos	`\arccos`	log	`\log`	lim sup	`\limsup`
	arcsin	`\arcsin`	lg	`\lg`	max	`\max`
	arctan	`\arctan`	arg	`\arg`	min	`\min`
	cosh	`\cosh`	deg	`\deg`	Pr	`\Pr`
	coth	`\coth`	gcd	`\gcd`		

Ces commandes mettent les noms de différentes fonctions mathématique en type romain, comme à l'accoutumée. Si vous appliquez une exposant ou un indice sur une de ces commandes, TEX le composera dans la plupart des cas à l'endroit usuel. En style d'affichage, TEX compose les exposants et les indices de `\det`, `\gcd`, `\inf`, `\lim`, `\liminf`, `\limsup`, `\max`, `\min`, `\Pr` et `\sup` comme s'ils étaient des limites, c'est-à-dire, directement au-dessus ou directement au-dessous du nom de la fonction.

Exemple :

```
$\cos^2 x + \sin^2 x = 1\qquad\max_{a \in A} g(a) = 1$
```

produit :

$\cos^2 x + \sin^2 x = 1 \qquad \max_{a \in A} g(a) = 1$

`\bmod`

Cette commande produit une opération binaire pour indiquer un modulo dans une formule.

Exemple :

```
$$x = (y+1) \bmod 2$$
```

produit :

$$x = (y + 1) \bmod 2$$

`\pmod`

Cette commande procure une notation pour indiquer un modulo entre parenthèses à la fin d'une formule.

Exemple :

```
$$x \equiv y+1 \pmod 2$$
```

produit :

$$x \equiv y + 1 \pmod 2$$

Grands opérateurs

☞	$\bigcap$	$\bigcap$	`\bigcap`	$\bigsqcup$	$\bigsqcup$	`\bigsqcup`	$\int$	`\smallint`
	$\bigcup$	$\bigcup$	`\bigcup`	$\biguplus$	$\biguplus$	`\biguplus`	$\int$	`\int`
	$\bigodot$	$\bigodot$	`\bigodot`	$\bigvee$	$\bigvee$	`\bigvee`	$\oint$	`\oint`
	$\bigoplus$	$\bigoplus$	`\bigoplus`	$\bigwedge$	$\bigwedge$	`\bigwedge`	$\prod$	`\prod`
	$\bigotimes$	$\bigotimes$	`\bigotimes`	$\coprod$	$\coprod$	`\coprod`	$\sum$	`\sum`

Ces commandes produisent différents grands symboles d'opérateur. TEX produit la plus petite taille quand il est en style texte et les plus grandes taille quand il est en style d'affichage. Les Opérateurs sont une des classes de symboles mathématiques de TEX. TEX mets différents montants d'espace autour des différentes classes de symboles mathématiques.

Les grand symboles d'opérateur avec '`big`' dans leurs noms sont différents des opérations binaires correspondantes (voir p. 197) comme `\cap` ($\cap$) puisqu'il apparaissent habituellement au début d'une formule. TEX utilise des espacements différents pour un grand opérateur et pour une opération binaire.

Ne confondez pas '$\sum$' (`\sum`) avec 'Σ'(`\Sigma`) ou '$\prod$' (`\prod`) avec 'Π' (`\Pi`). `\Sigma` et `\Pi` produisent les lettres grecques capitales, qui sont plus petite et ont une apparence différente.

Un grand opérateur peut avoir des limites. La limite la plus basse est spécifiée comme un indice et la plus haute comme un exposant.

Exemple :

```
$$\bigcap_{k=1}^r (a_k \cup b_k)$$
```

produit :

$$\bigcap_{k=1}^{r} (a_k \cup b_k)$$

Exemple :

```
$${\int_0^\pi \sin^2 ax\,dx} = {\pi \over 2}$$
```

produit :

$$\int_0^\pi \sin^2 ax\,dx = \frac{\pi}{2}$$

`\limits`

Quand il est en style texte, TEX place normalement les limites après un grand opérateur. Cette commande demande à TEX de placer les limites sur et sous un grand opérateur plutôt qu'après lui.

Si vous spécifiez plusieurs commandes `\limits`, `\nolimits`, et `\displaylimits`, seule la dernière sera prise en compte.

Exemple :

```
Suppose that $\bigcap\limits_{i=1}^Nq_i$ contains at least
two elements.
```

produit :

Suppose that $\bigcap\limits_{i=1}^{N} q_i$ contains at least two elements.

`\nolimits`

Quand il est en style d'affichage, TEX place normalement les limites sur et sous un grand opérateur. (L'opérateur `\int` est une exception—TEX places les limites pour `\int` après l'opérateur dans tous les cas.) Cette commande demande à TEX de placer les limites après un grand opérateur plutôt que sur et sous lui.

Si vous spécifiez plusieurs commandes `\limits`, `\nolimits`, et `\displaylimits`, seule la dernière sera prise en compte.

Exemple :

```
$$\bigcap\nolimits_{i=1}^Nq_i$$
```

produit :

$$\bigcap\nolimits_{i=1}^{N} q_i$$

`\displaylimits`

Cette commande demande à TEX de suivre ses règles normales pour le placement des limites :

1) Les limites de `\int` sont placées après l'opérateur.
2) Les limites des autres grands opérateurs sont placées après l'opérateur en style texte.
3) Les limites des autres grands opérateurs sont placées sur et sous l'opérateur en style d'affichage.

Il est habituellement plus simple d'utiliser `\limits` ou `\nolimits` pour produire un effet spécifique, mais `\displaylimits` est parfois utile dans les définitions de macro.

Notez que plain TEX définit `\int` comme une macro qui comprend `\nolimits`, donc `\int\displaylimits` en style de texte restaure la convention de `\limits`.

Si vous spécifiez plusieurs commandes `\limits`, `\nolimits`, et `\displaylimits`, seule la dernière sera prise en compte.

Exemple :

```
$$a(\lambda) = {1 \over {2\pi}} \int\displaylimits
_{-\infty}^{+\infty} f(x)e^{-i\lambda x}\,dx$$
```

produit :

$$a(\lambda) = \frac{1}{2\pi} \int\limits_{-\infty}^{+\infty} f(x)e^{-i\lambda x}\,dx$$

Ponctuation

`\cdotp`
`\ldotp`

Ces deux commandes produisent respectivement un point centré et un point positionné sur la ligne de base. Elles ne sont valide q'en mode mathématique. TEX les traite comme de la ponctuation, ne mettant aucun espace supplémentaire devant elles mais en ajoutant un petit en plus après elles. En contrepartie, TEX mets un montant d'espace égal des deux cotés d'un point centré généré par la commande `\cdot` (p. 197).

Exemple :

```
$x \cdotp y \quad x \ldotp y \quad x \cdot y$
```

produit :

$x \cdotp y \quad x \ldotp y \quad x \cdot y$

`\colon`

Cette commande produit une symbole de ponctuation deux-points. Elle n'est valide qu'en mode mathématique. La différence entre `\colon` et le caractère deux-points (`:`) est que '`:`' est un opérateur, donc TEX mets une espace supplémentaire à sa gauche tandis qu'il ne met pas d'espace supplémentaire à la gauche de `\colon`.

Exemple :

```
$f \colon t \quad f : t$
```

produit :

$f \colon t \quad f : t$

Puissances et indices

☞ `_` ⟨*argument*⟩ `^` ⟨*argument*⟩
`\sb` ⟨*argument*⟩ `\sp` ⟨*argument*⟩

Les commandes de chaque colonne sont équivalentes. Les commandes de la première colonne composent ⟨*argument*⟩ comme une puissance et celle de la seconde colonne composent ⟨*argument*⟩ comme un indice. Les commandes `\sb` et `\sp` sont principalement utiles si vous travaillez sur un terminal sur lequel manque un signe souligné ou de puissance ou si vous avez redéfini '`_`' or '`^`' et avez besoin d'accéder à sa définition originale. Ces commandes sont aussi utilisées pour mettre des limites inférieures ou supérieures sur des sommes ou des intégrales.

Si une puissance ou un indice n'est pas un token seul, vous devez l'englober dans un groupe. TEX ne mets pas de priorité aux puissances ou aux indices, donc il rejettera des formules telles que `a_i_j`, `a^i^j` ou `a^i_j`.

les puissances et les indices sont normalement composé en style script ou en style scriptscript s'il sont de second ordre, c'est-à-dire, un indice d'un indice ou une puissance d'une puissance. Vous pouvez mettre *tout* texte d'une formule mathématique en stylescript ou scriptscript avec les commandes `\scriptstyle` et `\scriptscriptstyle` (p. 206).

Vous pouvez appliquer une puissance ou un indice à tous les commandes qui produisent des fonction mathématique nommées en caractère romain (voir p. 201). Dans certain cas (voyez encore p. 201) la puissance ou l'indice apparaît directement sur ou sous la fonction nommée comme montré dans les exemples de `\lim` et de `\det` ci-dessous.

Exemple :

```
$x_3 \quad t_{\max} \quad a_{i_k} \quad \sum_{i=1}^n{q_i}
   \quad x^3\quad e^{t \cos\theta}\quad r^{x^2}\quad
   \int_0^\infty{f(x)\,dx}$
$$\lim_{x\leftarrow0}f(x)\qquad\det^{z\in A}\qquad\sin^2t$$
```

produit :

$x_3 \quad t_{\max} \quad a_{i_k} \quad \sum_{i=1}^n q_i \quad x^3 \quad e^{t\cos\theta} \quad r^{x^2} \quad \int_0^\infty f(x)\,dx$

$$\lim_{x\leftarrow 0} f(x) \qquad \det^{z\in A} \qquad \sin^2 t$$

Choisir et utiliser des modèles

`\textstyle`
`\scriptstyle`
`\scriptscriptstyle`
`\displaystyle`

Ces commandes forcent le style normal et par conséquent la police que TEX utilise pour composer une formule. comme les commandes de choix de police telles que `\it`, elle sont effective jusqu'a la fin du groupe les contenant. Elles sont utiles quand le choix de style de TEX est inapproprié pour la formule que vous essayez de composer.

Exemple :

```
$t+{\scriptstyle t + {\scriptscriptstyle t}}$
```

produit :

$t + {\scriptstyle t + {\scriptscriptstyle t}}$

`\mathchoice {` ⟨*math*$_1$⟩ `} {` ⟨*math*$_2$⟩ `} {` ⟨*math*$_3$⟩ `} {` ⟨*math*$_4$⟩ `}`

Cette commande demande à TEX de composer une des sous-formules ⟨*math*$_1$⟩, ⟨*math*$_2$⟩, ⟨*math*$_3$⟩ ou ⟨*math*$_4$⟩, en faisant son choix selon le style courant. Ainsi, si TEX est en style d'affichage il compose `\mathchoice` comme ⟨*math*$_1$⟩ ; en style texte il la compose comme ⟨*math*$_2$⟩ ; en style script il la compose comme ⟨*math*$_3$⟩ ; et en style scriptscript il la compose comme ⟨*math*$_4$⟩.

Exemple :

```
\def\mc{{\mathchoice{D}{T}{S}{SS}}}
The strange formula $\mc_{\mc_\mc}$ illustrates a
mathchoice.
```

produit :

The strange formula $T_{S_{SS}}$ illustrates a mathchoice.

`\mathpalette` ⟨*argument*$_1$⟩ ⟨*argument*$_2$⟩

Cette commande procure un moyen simple de produire une construction mathématique qui marche dans les quatre styles. Pour l'utiliser, vous

devrez normalement définir une macro supplémentaire, que nous appellerons `\build`. L'appel de `\mathpalette` pourra alors avoir la forme `\mathpalette\build`⟨*argument*⟩.

`\build` teste dans quel style est TEX et compose ⟨*argument*⟩ en conséquence. Il peut être défini pour avoir deux paramètres. Quand vous appelez `\mathpalette`, il appellera à son tout `\build`, avec `#1` une commande qui sélectionne le style courant et `#2` un ⟨*argument*⟩. Ainsi, dans la définition de `\build` vous pouvez composer quelque chose dans le style courant en le faisant précéder de '`#1`'. Voir la page 360 de *The TEXbook* et 999 de la traduction française pour des exemples d'utilisation de `\mathpalette` et la page 151 de *The TEXbook* et 999 de la traduction française pour plus d'explication sur son fonctionnement.

Symboles composés

Accents mathématiques

☞ `\acute` accent aigu comme dans $\acute{x}$
`\b` accent barre dessous comme dans $\underline{x}$
`\bar` accent barre comme dans $\bar{x}$
`\breve` accent bref comme dans $\breve{x}$
`\check` accent tchèque comme dans $\check{x}$
`\ddot` accent double point comme dans $\ddot{x}$
`\dot` accent point comme dans $\dot{x}$
`\grave` accent grave comme dans $\grave{x}$
`\hat` accent circonflexe comme dans $\hat{x}$
`\widehat` accent circonflexe large comme dans $\widehat{x+y}$
`\tilde` accent tilde comme dans $\tilde{x}$
`\widetilde` accent tilde large comme dans $\widetilde{z+a}$
`\vec` accent vecteur comme dans $\vec{x}$

Ces commandes produisent des accents dans les formules mathématiques. vous devrez normalement laisser un espace après chacune d'entre elles. Un accent large peut s'appliquer à une sous-formule comportant plusieurs caractères ; TEX centrera l'accent sur la sous-formule. Les autres accents ne sont applicable qu'a des caractères simples.

Exemple :

```
$\dot t^n \qquad \widetilde{v_1 + v_2}$
```

produit :

$\dot{t}^n \qquad \widetilde{v_1 + v_2}$

`\mathaccent` ⟨*mathcode*⟩

Ces commande demandent à TEX de composer un accent mathématique dont la famille et le code de caractère sont donné par ⟨*mathcode*⟩. (TEX

ignore la classe du mathcode.) Voir l'Annexe G de *The TEXbook* et de la traduction française pour les détails sur la manière dont TEX positionne de tels accents. La façon usuelle d'utiliser `\mathaccent` est de le mettre dans une définition de macro que donne un nom à l'accent mathématique.

Exemple :

```
\def\acute{\mathaccent "7013}
```

Voir aussi : "Accents" (p. 106).

Fractions et autres opérations empilées

☞ `\over`
`\atop`
`\above` ⟨*dimension*⟩
`\choose`
`\brace`
`\brack`

Ces commandes empile une sous-formule sur une autre. Nous expliquerons comment marche `\over` et lui relierons ensuite les autres commandes.

`\over` est la commande que vous utilisez normalement pour produire une fraction. Si vous écrivez quelque chose sous une des formes suivantes :

`$$`⟨*formula*$_1$⟩`\over`⟨*formula*$_2$⟩`$$`
`$`⟨*formula*$_1$⟩`\over`⟨*formula*$_2$⟩`$`
`\left`⟨*delim*⟩⟨*formula*$_1$⟩`\over`⟨*formula*$_2$⟩`\right`⟨*delim*⟩
`{`⟨*formula*$_1$⟩`\over`⟨*formula*$_2$⟩`}`

vous obtiendrez une fraction avec un numérateur ⟨*formula*$_1$⟩ et un dénominateur ⟨*formula*$_2$⟩, c'est-à-dire, ⟨*formula*$_1$⟩ sur ⟨*formula*$_2$⟩. Dans la première de ces trois formes, le `\over` n'est pas implicitement contenu dans un groupe ; il absorbe tout ce qui est à sa gauche et à sa droite jusqu'a ce qu'il trouve une frontière, à savoir, le début ou la fin d'un groupe.

Vous ne pouvez pas utiliser `\over` ou un autre de ces commandes de ce groupe plus d'une fois dans une formule. Ainsi une formule telle que :

```
$$a \over n \choose k$$
```

n'est pas légale. Ce n'est pas une sévère restriction parce que vous pouvez toujours englober une de ces commandes entre accolades. La raison de cette restriction est que si vous avez deux de ces commandes dans une seule formule, TEX ne saura pas comment les grouper.

Les autres commandes sont similaire à `\over`, avec les exceptions suivantes :

- `\atop` omet la barre de fraction.
- `\above` procure une barre de fraction d'épaisseur ⟨*dimension*⟩.
- `\choose` omet la barre de fraction et englobe la construction entre parenthèses. (Elle est appelée "choose" parce que $\binom{n}{k}$ est la notation pour le nombre de façons de choisir k choses parmi n.)

- `\brace` omet la barre de fraction et englobe la construction entre accolades.
- `\brack` omet la barre de fraction et englobe la construction entre crochets.

Exemple :

```
$${n+1 \over n-1}      \qquad {n+1 \atop n-1}   \qquad
  {n+1 \above 2pt n-1} \qquad {n+1 \choose n-1} \qquad
  {n+1 \brace n-1}     \qquad {n+1 \brack n-1}$$
```

produit :

$$\frac{n+1}{n-1} \qquad n+1 \atop n-1 \qquad \frac{n+1}{n-1} \qquad \binom{n+1}{n-1} \qquad \left\{{n+1 \atop n-1}\right\} \qquad \left[{n+1 \atop n-1}\right]$$

`\overwithdelims` ⟨*delim*$_1$⟩ ⟨*delim*$_2$⟩
`\atopwithdelims` ⟨*delim*$_1$⟩ ⟨*delim*$_2$⟩
`\abovewithdelims` ⟨*delim*$_1$⟩ ⟨*delim*$_2$⟩ ⟨*dimension*⟩

Chacune de ces commandes superpose une sous-formule sur une autre et entoure le résultat avec ⟨*delim*$_1$⟩ à gauche et ⟨*delim*$_2$⟩ à droite. Ces commandes suivent les même règles que `\over`, `\atop` et `\above`. Le ⟨*dimen*⟩ dans `\abovewithdelims` spécifie l'épaisseur de la barre de fraction.

Exemple :

```
$${m \overwithdelims () n}\qquad
  {m \atopwithdelims || n}\qquad
  {m \abovewithdelims \{\} 2pt n}$$
```

produit :

$$\left(\frac{m}{n}\right) \qquad \left|{m \atop n}\right| \qquad \left\{\frac{m}{n}\right\}$$

`\cases`

Cette commande produit la forme mathématique qui décrit un choix entre plusieurs cas. Chaque cas a deux parties, séparées par '`&`'. TEX traite la première partie comme une formule mathématique et la seconde partie comme du texte ordinaire. Chaque cas doit être suivi par `\cr`.

Exemple :

```
$$g(x,y) = \cases{f(x,y),&if $x<y$\cr
                  f(y,x),&if $x>y$\cr
                  0,&otherwise.\cr}$$
```

produit :

$$g(x,y) = \begin{cases} f(x,y), & \text{if } x < y \\ f(y,x), & \text{if } x > y \\ 0, & \text{otherwise.} \end{cases}$$

`\underbrace` ⟨*argument*⟩
`\overbrace` ⟨*argument*⟩
`\underline` ⟨*argument*⟩
`\overline` ⟨*argument*⟩
`\overleftarrow` ⟨*argument*⟩
`\overrightarrow` ⟨*argument*⟩

Ces commandes placent des accolades, des lignes ou des flèches extensible sur ou sous la sous-formule donnée par ⟨*argument*⟩. TEX fera cette construction aussi large que nécessaire pour le contexte. Quand TEX produit les accolades, lignes ou flèches étendues, il ne considère que les dimensions de la boîte contenant ⟨*argument*⟩. Si vous utilisez plus d'une de ces commandes dans une seule formule, les accolades, lignes ou flèches qu'il produira pourront ne pas s'aligner proprement les unes avec les autres. Vous pouvez utiliser la commande `\mathstrut` (p. 174) pour contourner cette difficulté.

Exemple :

```
$$\displaylines{
\underbrace{x \circ y}\qquad \overbrace{x \circ y}\qquad
\underline{x \circ y}\qquad \overline{x \circ y}\qquad
\overleftarrow{x \circ y}\qquad
\overrightarrow{x \circ y}\cr
{\overline r + \overline t}\qquad
{\overline {r \mathstrut} + \overline {t \mathstrut}}\cr
}$$
```

produit :

$$\underbrace{x \circ y} \qquad \overbrace{x \circ y} \qquad \underline{x \circ y} \qquad \overline{x \circ y} \qquad \overleftarrow{x \circ y} \qquad \overrightarrow{x \circ y}$$

$$\overline{r} + \overline{t} \qquad \overline{\overline{r} + \overline{t}}$$

`\buildrel` ⟨*formule*⟩ `\over` ⟨*relation*⟩

Cette commande produit une boîte dans laquelle ⟨*formula*⟩ est placé sur ⟨*relation*⟩. TEX traite le résultat comme une relation en ce qui concerne l'espacement (voir "classe", p. 56).

Exemple :

```
$\buildrel \rm def \over \equiv$
```

produit :
$\stackrel{\text{def}}{\equiv}$

Points

☞ `\ldots`
`\cdots`
Ces commandes produisent trois points alignés. Pour `\ldots`, les points sont sur la ligne de base ; pour `\cdots`, les points sont centrés en respectant les axe (voir l'explication de `\vcenter`, p. 221).

Exemple :
```
$t_1 + t_2 + \cdots + t_n \qquad x_1,x_2, \ldots\,, x_r$
```
produit :
$t_1 + t_2 + \cdots + t_n \qquad x_1, x_2, \ldots, x_r$

☞ `\vdots`
Cette commande produit trois points verticaux.

Exemple :
```
$$\eqalign{f(\alpha_1)& = f(\beta_1)\cr
   \noalign{\kern -4pt}%
   &\phantom{a}\vdots\cr % moves the dots right a bit
   f(\alpha_k)& = f(\beta_k)\cr}$$
```
produit :
$$\begin{aligned} f(\alpha_1) &= f(\beta_1) \\ &\vdots \\ f(\alpha_k) &= f(\beta_k) \end{aligned}$$

`\ddots`
Cette commande produit trois points sur une diagonale. Son usage le plus commun est d'indiquer une répétition le long de la diagonale d'une matrice.

Exemple :
```
$$\pmatrix{0&\ldots&0\cr
           \vdots&\ddots&\vdots\cr
           0&\ldots&0\cr}$$
```
produit :
$$\begin{pmatrix} 0 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & 0 \end{pmatrix}$$

Voir aussi : `\dots` (p. 105).

Délimiteurs

`\lgroup`
`\rgroup`
Ces commandes produisent de grandes parenthèses gauches et droites qui sont définies comme des délimiteurs ouvrants et fermants. La plus petite taille disponible pour ces délimiteurs est `\Big`. Si vous utilisez des tailles plus petites, vous obtiendrez des caractères étranges.

Exemple :

```
$$\lgroup\dots\rgroup\qquad\bigg\lgroup\dots\bigg\rgroup$$
```

produit :

$$(\dots) \qquad \left(\dots\right)$$

☞ `\left`
`\right`
Ces commandes doivent être employées ensemble selon le modèle :

`\left` ⟨*delim*$_1$⟩ ⟨*subformula*⟩ `\right` ⟨*delim*$_2$⟩

Cette construction demande à TEX de produire ⟨*subformula*⟩, entourée des délimiteurs ⟨*delim*$_1$⟩ et ⟨*delim*$_2$⟩. La taille verticale des délimiteur est ajustée pour correspondre à la taille vertical (hauteur plus profondeur) de ⟨*subformula*⟩. ⟨*delim*$_1$⟩ et ⟨*delim*$_2$⟩ n'ont pas besoin de correspondre. Par exemple, vous pouvez utiliser '`]`' comme d'un délimiteur gauche et '`(`' comme d'un délimiteur droite dans une utilisation simple de `\left` et `\right`.

`\left` et `\right` ont la propriété importante de définir un groupe, c'est-à-dire, qu'ils agissent comme des accolades gauches et droite. cette propriété de groupement est particulièrement utile quand vous mettez `\over` (p. 208) ou une commande similaire entre `\left` et `\right`, car vous n'avez pas besoin de mettre d'accolades autour de la fraction construite par `\over`.

Si vous voulez un délimiteur gauche mais pas le droit, vous pouvez utiliser '`.`' à la place du délimiteur que vous ne voulez pas et il se transformera en espace blanc (de largeur `\nulldelimiterspace`).

Exemple :

```
$$\left\Vert\matrix{a&b\cr c&d\cr}\right\Vert
  \qquad \left\uparrow q_1\atop q_2\right.$$
```

produit :

$$\left\Vert\begin{matrix}a&b\\c&d\end{matrix}\right\Vert \qquad \left\uparrow\begin{matrix}q_1\\q_2\end{matrix}\right.$$

`\delimiter` ⟨*nombre*⟩
Cette commande produit un délimiteur dont les caractéristiques sont données par ⟨*number*⟩. ⟨*number*⟩ est normalement écrit en notation hexadécimale. Vous pouvez utiliser la commande `\delimiter` à la place d'un caractère dans tout contexte où TEX attend un délimiteur (bien que la commande soit rarement employée en dehors d'une définition de macro). Supposez que ⟨*number*⟩ soit le nombre hexadécimal $cs_1s_2s_3l_1l_2l_3$. Alors TEX prendra le délimiteur ayant la classs c, la petite variante $s_1s_2s_3$ et la grande variante $l_1l_2l_3$. Ici $s_1s_2s_3$ indique le caractère mathématique trouvé en position s_2s_3 de la famille s_1 et de même pour $l_1l_2l_3$. C'est la même convention que celle utilisée pour `\mathcode` (p. 259).

Exemple :

```
\def\vert{\delimiter "026A30C} % As in plain TeX.
```

`\delimiterfactor` [⟨*nombre*⟩ paramètre]
`\delimitershortfall` [⟨*nombre*⟩ paramètre]
Ces deux paramètres signalent à TEX comment la hauteur d'un délimiteur doit être reliée à la taille verticale de la sous-formule à laquelle le délimiteur est associé. `\delimiterfactor` donne le ratio minimum de la taille du délimiteur par rapport à la taille verticale de la sous-formule et `\delimitershortfall` donne le maximum par lequel la hauteur du délimiteur sera réduit pour correspondre à la taille verticale de la sous-formule.

Supposez que la boîte contenant la sous-formule ait une hauteur h et une profondeur d et $y = 2\max(h, d)$. soit la valeur de `\delimiterfactor` f et la valeur de `\delimitershortfall` δ. Alors TEX prend la taille de délimiteur minimum étant au moins $y \cdot f/1000$ et au moins $y - \delta$. En particulier, si `\delimiterfactor` est exactement à 1000 alors TEX essayera de faire un délimiteur au moins aussi grand que la formule à laquelle il est attaché. Voir la page 152 et la page 446 (Règle 19) de *The TEXbook* et page 999 et la page 999 (Règle 19) de la traduction française pour les détail exactes sur la façon dont TEX utilise ces paramètres. Plain TEX mets `\delimiterfactor` à 901 et `\delimitershortfall` à `5pt`.

Voir aussi : `\delcode` (p. 260), `\vert`, `\Vert` et `\backslash` (p. 196).

Matrices

`\matrix {` ⟨*ligne*⟩ `\cr` ... ⟨*ligne*⟩ `\cr }`
`\pmatrix {` ⟨*ligne*⟩ `\cr` ... ⟨*ligne*⟩ `\cr }`
`\bordermatrix {` ⟨*ligne*⟩ `\cr` ... ⟨*ligne*⟩ `\cr }`
Chacune de ces trois commandes produit une matrice. Les éléments de chaque rangé de la matrice entrée sont séparés par '`&`' et chaque rangé est à son tour terminée par `\cr`. (C'est la même forme que celle utilisée pour un alignement.) Les commandes diffèrent de manières suivantes :

- `\matrix` produit une matrice sans aucun entourage ou délimiteurs insérés.
- `\pmatrix` produit une matrice entourée par des parenthèses.
- `\bordermatrix` produit une matrix dans laquelle la première rangé et la première colonne sont traitées comme des labels. (Le premier élément de la première rangé est habituellement laissé à blanc.) Le reste de la matrice est englobé dans des parenthèses.

TEX peut rendre les parenthèses de `\pmatrix` et de `\bordermatrix` aussi large que nécessaire en insérant des extensions verticales. Si vous voulez une matrice entourée par des délimiteurs autres que des parenthèses, vous devez utiliser `\matrix` en conjugaison avec `\left` et `\right` (p. 212).

Exemple :

```
$$\displaylines{
   \matrix{t_{11}&t_{12}&t_{13}\cr
           t_{21}&t_{22}&t_{23}\cr
           t_{31}&t_{32}&t_{33}\cr}\qquad
\left\{\matrix{t_{11}&t_{12}&t_{13}\cr
           t_{21}&t_{22}&t_{23}\cr
           t_{31}&t_{32}&t_{33}\cr}\right\}\cr
\pmatrix{t_{11}&t_{12}&t_{13}\cr
           t_{21}&t_{22}&t_{23}\cr
           t_{31}&t_{32}&t_{33}\cr}\qquad
\bordermatrix{&c_1&c_2&c_3\cr
           r_1&t_{11}&t_{12}&t_{13}\cr
           r_2&t_{21}&t_{22}&t_{23}\cr
           r_3&t_{31}&t_{32}&t_{33}\cr}\cr}$$
```

produit :

$$\begin{matrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{matrix} \qquad \left\{\begin{matrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{matrix}\right\}$$

$$\begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{pmatrix} \qquad \begin{matrix} & c_1 & c_2 & c_3 \\ r_1 & t_{11} & t_{12} & t_{13} \\ r_2 & t_{21} & t_{22} & t_{23} \\ r_3 & t_{31} & t_{32} & t_{33} \end{matrix}$$

Racines et radicaux

☞ `\sqrt` ⟨*argument*⟩

Cette commande produit la notation pour la racine carrée de ⟨*argument*⟩.

Exemple :

```
$$x = {-b\pm\sqrt{b^2-4ac} \over 2a}$$
```

produit :

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

☞ `\root` ⟨*argument*$_1$⟩ `\of` ⟨*argument*$_2$⟩
Cette commande produit la notation pour un racine de ⟨*argument*$_2$⟩ où l'ordre est donné par ⟨*argument*$_1$⟩.

Exemple :
```
$\root \alpha \of {r \cos \theta}$
```
produit :
$\sqrt[\alpha]{r \cos \theta}$

`\radical` ⟨*nombre*⟩
Cette commande produit un signe radical dont les caractéristiques sont données par ⟨*nombre*⟩. Elle utilise la même représentation que le code délimiteur dans la commande `\delcode` (p. 260).

Exemple :
```
\def\sqrt{\radical "270370} % as in plain TeX
```

Numéros d'équation

☞ `\eqno`
`\leqno`
Ces commandes attachent un numéro d'équation à une formule affichée. `\eqno` mets le numéro d'équation à droite et `\leqno` le mets à gauche. Les commandes doivent être mises à la fin de la formule. Si vous avez un affichage multi-ligne et voulez numéroter plus d'une ligne, utilisez les commandes `\eqalignno` ou `\leqalignno` (p. 216).

Ces commandes ne sont valides qu'en mode mathématique affiché.

Exemple :
```
$$e^{i\theta} = \cos \theta + i \sin \theta\eqno{(11)}$$
```
produit :

$$e^{i\theta} = \cos \theta + i \sin \theta \tag{11}$$

Exemple :
```
$$\cos^2 \theta + \sin^2 \theta = 1\leqno{(12)}$$
```
produit :

$$\cos^2 \theta + \sin^2 \theta = 1 \tag{12}$$

Affichages multi-ligne

`\displaylines {` ⟨*ligne*⟩ `\cr`...⟨*ligne*⟩ `\cr }`
Cette commande produit un affichage mathématique multi-ligne dans lequel chaque ligne est centré indépendamment des autres lignes. Vous pouvez utiliser la commande `\noalign` (p. 189) pour changer le montant d'espace entre deux lignes d'un affichage multi-ligne.

Si vous voulez attacher des numéro d'équation à certaine ou toutes les équations dans un affichage mathématique multi-ligne, vous devez utiliser `\eqalignno` ou `\leqalignno`.

Exemple :

```
$$\displaylines{(x+a)^2 = x^2+2ax+a^2\cr
                (x+a)(x-a) = x^2-a^2\cr}$$
```

produit :

$$(x+a)^2 = x^2+2ax+a^2$$
$$(x+a)(x-a) = x^2-a^2$$

`\eqalign {` ⟨*ligne*⟩ `\cr` ... ⟨*ligne*⟩ `\cr }`
`\eqalignno {` ⟨*ligne*⟩ `\cr` ... ⟨*ligne*⟩ `\cr }`
`\leqalignno {` ⟨*ligne*⟩ `\cr` ... ⟨*ligne*⟩ `\cr }`
Ces commandes produisent un affichage mathématique multi-ligne dans lequel certaines parties correspondantes des lignes sont alignées verticalement. Les commandes `\eqalignno` et `\leqalignno` vous laissent aussi apporter des numéros d'équation pour certaines ou toutes les lignes. `\eqalignno` mets les numéros d'équation à droite et `\leqalignno` les mets à gauche.

Chaque ligne dans l'affichage est terminées par `\cr`. Chacune des parties devant être alignées (le plus souvent un signe égal) est précédé par '`&`'. Un '`&`' précède aussi caque numéro d'équation, qui est donné à la fin d'une ligne. Vous pouvez mettre plus d'une de ces commandes dans un seul affichage pour produire plusieurs groupes d'équation. Dans ce cas, seuls les groupes les plus à droites ou les plus à gauche peuvent être produit avec `\eqalignno` ou `\leqalignno`.

Vous pouvez utiliser la commande `\noalign` (p. 189) pour chager le montant d'espace entre deux lignes d'un affichage multi-ligne.

Exemple :

```
$$\left\{\eqalign{f_1(t) &= 2t\cr f_2(t) &= t^3\cr
          f_3(t) &= t^2-1\cr}\right\}
   \left\{\eqalign{g_1(t) &= t\cr g_2(t) &= 1}\right\}$$
```

produit :

$$\left\{\begin{array}{l} f_1(t) = 2t \\ f_2(t) = t^3 \\ f_3(t) = t^2 - 1 \end{array}\right\} \left\{\begin{array}{l} g_1(t) = t \\ g_2(t) = 1 \end{array}\right\}$$

Exemple :

```
$$\eqalignno{
\sigma^2&=E(x-\mu)^2&(12)\cr
   &={1 \over n}\sum_{i=0}^n (x_i - \mu)^2&\cr
   &=E(x^2)-\mu^2\cr}$$
```

produit :

$$\begin{aligned} \sigma^2 &= E(x-\mu)^2 && (12)\\ &= \frac{1}{n}\sum_{i=0}^{n}(x_i-\mu)^2 \\ &= E(x^2)-\mu^2 \end{aligned}$$

Exemple :

```
$$\leqalignno{
\sigma^2&=E(x-\mu)^2&(6)\cr
   &=E(x^2)-\mu^2&(7)\cr}$$
```

produit :

$$\begin{aligned} \sigma^2 &= E(x-\mu)^2 && (6)\\ &= E(x^2)-\mu^2 && (7) \end{aligned}$$

Exemple :

```
$$\eqalignno{
  &(x+a)^2 = x^2+2ax+a^2&(19)\cr
  &(x+a)(x-a) = x^2-a^2\cr}$$
% same effect as \displaylines but with an equation number
```

produit :

$$\begin{aligned} (x+a)^2 &= x^2+2ax+a^2 && (19)\\ (x+a)(x-a) &= x^2-a^2 \end{aligned}$$

Polices dans des formules mathématiques

☞ `\cal` utilise un police calligraphique majuscule
`\mit` utilise un police mathématique italique
`\oldstyle` utilise une police de chiffres elzéviriens
Ces commandes demande à TEX de composer le texte suivant dans la police spécifiée. Vous ne pouvez les utiliser qu'en mode mathématique.

La commande `\mit` est utile pour produire des lettres grecques capitales penchées. Vous pouvez aussi utiliser les commandes données dans "Sélectionner des polices" (p. 108) pour changer des polices en mode mathématique.

Exemple :

```
${\cal XYZ} \quad
{\mit AaBb\Gamma \Delta \Sigma} \quad
{\oldstyle 0123456789}$
```

produit :

$\mathcal{XYZ}$ $AaBb\Gamma\Delta\Sigma$ 0123456789

`\itfam` famille de type italique
`\bffam` famille de type grasse
`\slfam` famille de type penchée
`\ttfam` famille de type machine à écrire

Ces commandes définissent des familles de type devant être utilisées en mode mathématique. Leur principale utilisation est dans le définition des commandes `\it`, `\bf`, `\sl` et `\tt` pour qu'elles puissent fonctionner en mode mathématique.

`\fam` [⟨*nombre*⟩ paramètre]

Quand TEX est en mode mathématique, il compose normalement un caractère en utilisant la de famille de police donné dans son mathcode. Cependant, quand TEX est en mode mathématique et rencontre un caractère dont la classe est 7 (Variable), il compose ce caractère en utilisant la famille de police donnée par la valeur de `\fam`, à condition que la valeur de `\fam` soit entre 0 et 15. Si la valeur de `\fam` n'est pas dans cette interval, TEX utilise la famille du mathcode du caractère comme dans le cas ordinaire. TEX mets `\fam` à -1 à chaque fois qu'il entre en mode mathématique. En ddehors du mode mathématique, `\fam` n'a pas d'effet.

En assignant une valeur à `\fam` vous pouvez changer la façon dont TEX compose des caractères ordinaire tels que des variables. Par exemple, en mettant `\fam` à `\ttfam`, vous demandez à TEX de composer des variables en utilisant une police de machine à écrire. Plain TEX définit `\tt` comme une macro qui, entre autres choses, mets `\fam` à `\ttfam`.

Exemple :

```
\def\bf{\fam\bffam\tenbf} % As in plain TeX.
```

`\textfont` ⟨*family*⟩ [⟨*fontname*⟩ paramètre]
`\scriptfont` ⟨*family*⟩ [⟨*fontname*⟩ paramètre]
`\scriptscriptfont` ⟨*family*⟩ [⟨*fontname*⟩ paramètre]
Chacun de ces paramètres spécifie la police que TEX utilise pour composer le style indiqué dans la famille indiquée. Ces choix n'ont aucun effet en dehors du mode mathématique.

Exemple :

```
\scriptfont2 = \sevensy % As in plain TeX.
```

Voir aussi : "Type styles" (p. 109).

Construire des symboles mathématiques

Rendre des délimiteurs plus grand

☞

`\big`	`\Big`	`\bigg`	`\Bigg`
`\bigl`	`\Bigl`	`\biggl`	`\Biggl`
`\bigm`	`\Bigm`	`\biggm`	`\Biggm`
`\bigr`	`\Bigr`	`\biggr`	`\Biggr`

Ces commandes rendent des délimiteurs plus grand que leur taille normale. Les commandes dans les quatre colonnes produisent successivement de plus grandes tailles. La différence entre `\big`, `\bigl`, `\bigr` et `\bigm` est en relation avec la classe du délimiteur agrandi :

- `\big` produit un symbole ordinaire.
- `\bigl` produit un symbole ouvrant.
- `\bigr` produces un symbole fermant.
- `\bigm` produces un symbole de relation.

TEX utilise la classe d'un symbole pour décider combien d'espace mettre autour de ce symbole.

Ces commandes, contrairement à `\left` et `\right`, ne définissent *pas* un groupe.

Exemple :

```
$$(x) \quad \bigl(x\bigr) \quad \Bigl(x\Bigr) \quad
   \biggl(x\biggr) \quad \Biggl(x\Biggr)\qquad
[x] \quad \bigl[x\bigr] \quad \Bigl[x\Bigr] \quad
   \biggl[x\biggr] \quad \Biggl[x\Biggr]$$
```

produit :

$$(x) \quad (x) \quad \bigl(x\bigr) \quad \Bigl(x\Bigr) \quad \Biggl(x\Biggr) \qquad [x] \quad [x] \quad \bigl[x\bigr] \quad \Bigl[x\Bigr] \quad \Biggl[x\Biggr]$$

Parties de grands symboles

`\downbracefill`
`\upbracefill`
Ces commandes produisent respectivement des accolades horizontales et extensibles dirigées vers le haut et vers le bas. TEX rendra les accolades aussi larges que nécessaire. Ces commandes sont utilisées dans les définitions de `\overbrace` et de `\underbrace` (p. 210).

Exemple :
```
$$\hbox to 1in{\downbracefill} \quad
   \hbox to 1in{\upbracefill}$$
```
produit :

$$\underbrace{\hbox to 1in{}}\quad\overbrace{\hbox to 1in{}}$$

`\arrowvert`
`\Arrowvert`
`\lmoustache`
`\rmoustache`
`\bracevert`
Ces commandes produisent des portions de certain grand délimiteurs qui peuvent eux-même être utilisés comme délimiteurs. Elles font référence aux caractères de la police mathématique `cmex10`.

Exemple :
```
$$\cdots \Big\arrowvert \cdots \Big\Arrowvert \cdots
  \Big\lmoustache \cdots \Big\rmoustache \cdots
  \Big\bracevert \cdots$$
```
produit :

$$\cdots\Big|\cdots\Big\|\cdots\Big\lmoustache\cdots\Big\rmoustache\cdots\Big\bracevert\cdots$$

Aligner des parties d'une formule

Aligner des accents

`\skew` ⟨*nombre*⟩ ⟨*argument*$_1$⟩ ⟨*argument*$_2$⟩
Cette commande déplace l'accent ⟨*argument*$_1$⟩ de ⟨*number*⟩ unités mathématiques vers la droite de sa position normale en respectant ⟨*argument*$_2$⟩.

L'utilisation la plus commune de cette commande est pour modifier la position d'un accent qui se trouve placé sur un autre accent.

Exemple :

```
$$\skew 2\bar{\bar z}\quad\skew 3\tilde{\tilde y}\quad
  \skew 4\tilde{\hat x}$$
```

produit :

$$\bar{\bar{z}} \quad \tilde{\tilde{y}} \quad \tilde{\hat{x}}$$

`\skewchar` ⟨*font*⟩ [⟨*number*⟩ paramètre]

Le `\skewchar` d'une police est le caractère dans la police dont les crénages, défini dans le fichier de métrique de la police, détermine les positions des accents mathématiques. Cela dit, supposez que TEX applique un accent mathématique au caractère '`x`'. TEX regarde si la paire de caractère '`x\skewchar`' a un crénage ; si oui, il déplace l'accent d'un montant de ce crénage. l'algorithme complet que TEX utilise pour positionner les accents mathématiques (qui détermine beaucoup d'autres choses) se trouve dans l'annexe G de *The TEXbook* et de la traduction française.

Si la valeur de `\skewchar` n'est pas dans l'interval 0–255, TEX prend une valeur de crénage à zéro.

Notez que ⟨*font*⟩ est une séquence de contrôle qui nomme une police, pas un ⟨*nom de police*⟩ qui nomme des fichiers de police. Attention : un assignement de `\skewchar` n'est *pas* enlevé à la fin d'un groupe. Si vous voulez changer `\skewchar` localement, vous devrez sauvegarder et restaurer sa valeur original explicitement.

`\defaultskewchar` [⟨*number*⟩ paramètre]

Quand TEX lit le fichier de métriques pour une police en réponse à une commande `\font`, il mets le `\skewchar` de la police dans `\defaultskewchar`. Si la valeur de `\defaultskewchar` n'est pas dans l'interval 0–255, TEX n'assigne aucun caractère oblique par défaut. Plain TEX mets `\defaultskewchar` à −1, et il est normalement préférable de le laisser ainsi.

Aligner du matériel verticalement

`\vcenter {` ⟨*matériel en mode vertical*⟩ `}`
`\vcenter to` ⟨*dimension*⟩ `{` ⟨*matériel en mode vertical*⟩ `}`
`\vcenter spread` ⟨*dimension*⟩ `{` ⟨*matériel en mode vertical*⟩ `}`

Toutes les formules mathématiques ont un "axe" invisible que TEX traite comme une sorte de ligne de centrage horizontale pour cette formule. Par exemple, l'axe d'une formule constituée d'une fraction est au centre de la barre de fraction. La commande `\vcenter` demande à TEX de placer le ⟨*matériel en mode vertical*⟩ dans une vbox et de centrer la vbox en respectant l'axes de la formule qu'il est en train de construire.

La première forme de la commande centre le matériel comme donné. Les deuxièmes et troisièmes forme élargissent ou rétrécissent le matériel verticalement comme dans la commande `\vbox` (p. 167).

Exemple :

```
$${n \choose k} \buildrel \rm def \over \equiv \>
\vcenter{\hsize 1.5 in \noindent the number of
combinations of $n$ things taken $k$ at a time}$$
```

produit :

$$\binom{n}{k} \stackrel{\text{def}}{\equiv} \text{the number of combinations of } n \text{ things taken } k \text{ at a time}$$

Produire des espaces

Espaces mathématiques de largeur fixe

```
\!
\,
\>
\;
```

Ces commandes produisent des montants variés d'espace supplémentaire dans des formules. Elles sont définies en termes d'unités mathématiques, donc TEX ajuste le montant d'espace en accord avec le style courant.

- `\!` produit un espace fin négatif, c'est-à-dire, qu'il réduit l'espace entre ses sous-formules voisines du montant d'un espace fin.
- `\,` produit un espace fin.
- `\>` produit un espace moyen.
- `\;` produit un espace large.

Exemple :

```
$$00\quad0\!0\quad0\,0\quad0\>0\quad0\;0\quad
{\scriptstyle 00\quad0\!0\quad0\,0\quad0\>0\quad0\;0}$$
```

produit :

$$00 \quad 00 \quad 0\,0 \quad 0\,0 \quad 0\,0 \quad {\scriptstyle 00} \quad {\scriptstyle 00} \quad {\scriptstyle 0\,0} \quad {\scriptstyle 0\,0} \quad {\scriptstyle 0\,0}$$

`\thinmuskip` [⟨*muglue*⟩ paramètre]
`\medmuskip` [⟨*muglue*⟩ paramètre]
`\thickmuskip` [⟨*muglue*⟩ paramètre]

Ces paramètres définissent des espaces fins, moyens et larges en mode mathématique.

Exemple :

```
$00\quad0\mskip\thinmuskip0\quad0\mskip\medmuskip0
   \quad0\mskip\thickmuskip0$
```

produit :

00 00 0 0 0 0

`\jot` [⟨*dimension*⟩ paramètre]

Ce paramètre défini une distance égale à trois points (à moins que vous le changiez). Le `\jot` est une unité de mesure pratique pour ouvrir des affichages mathématiques.

Espaces mathématiques de largeur variable

`\mkern` ⟨*mudimen*⟩

Cette commande produit un crénage, c'est-à-dire, un espace blanc, de largeur ⟨*mudimen*⟩. Le crénage est mesuré en unités mathématiques, qui varie en fonction du style. Hormis son unité de la mesure, cette commande se comporte comme `\kern` (p. 163) le fait en mode horizontal.

Exemple :

```
$0\mkern13mu 0 \qquad {\scriptscriptstyle 0 \mkern13mu 0}$
```

produit :

0 0 0 0

`\mskip` ⟨*mudimen*$_1$⟩ `plus` ⟨*mudimen*$_2$⟩ `minus` ⟨*mudimen*$_3$⟩

Cette commande produit un ressort horizontal qui a une largeur naturelle ⟨*mudimen*$_1$⟩, s'étire de ⟨*mudimen*$_2$⟩ et se rétrécit de ⟨*mudimen*$_3$⟩. Le ressort est mesuré en unités mathématiques, qui varie en fonction du style. Hormis son unité de mesure, cette commande se comporte comme `\hskip` (p. 161).

Exemple :

```
$0\mskip 13mu 0 \quad {\scriptscriptstyle 0 \mskip 13mu 0}$
```

produit :

0 0 0 0

`\nonscript`

Quand TEX est en train de composer en style script ou scriptscript et rencontre cette commande immédiatement devant d'un ressort ou d'un crénage, il supprime le ressort ou le crénage. `\nonscript` n'a pas d'effet dans les autres styles.

Cette commande fournit une manière de "resserrer" l'espacement dans les styles script et scriptscript, qui sont généralement faits dans une police plus petite. Elle est peu utile en dehors des définitions de macro.

Exemple :

```
\def\ab{a\nonscript\; b}
$\ab^{\ab}$
```

produit :

$a\ b^{a\ b}$

Voir aussi : `\kern` (p. 163), `\hskip` (p. 161).

paramètres d'espacement pour les affichages

`\displaywidth` [⟨*dimension*⟩ paramètre]

Ce paramètre spécifie la largeur maximum que TEX alloue pour un affichage mathématique. Si TEX ne peut pas mettre l'affichage dans un espace de cette largeur, il fait un "overfull hbox" et se plaint. TEX met la valeur de `\displaywidth` quand il rencontre le '`$$`' qui débute l'affichage. Cette valeur initiale est `\hsize` (p. 120) à moins qu'elle soit modifiée par des changements de la forme du paragraphe. Voir les pages 188–189 de *The TEXbook* et 999–999 de la traduction française pour une explication plus détaillée de ce paramètre.

`\displayindent` [⟨*dimension*⟩ paramètre]

Ce paramètre spécifie l'espace par lequel TEX indente un affichage mathématique. TEX met la valeur de `\displayindent` quand il rencontre le '`$$`' qui débute l'affichage. Normalement la valeur initiale est zéro, mais si la forme du paragraphe indique que l'affichage doit être décalé d'un montant s, TEX mettra `\displayindent` à s. Voir les pages 188–189 de *The TEXbook* et 999–999 de la traduction française pour une explication plus détaillée de ce paramètre.

`\predisplaysize` [⟨*dimension*⟩ paramètre]

TEX met ce paramètre à la largeur de la ligne précédant un affichage mathématique. TEX utilise `\predisplaysize` pour déterminer si l'affichage débute ou non à gauche d'où la ligne précédente se termine, c'est-à-dire, s'il recouvre visuellement ou non la ligne précédente. S'il la recouvre, il utilise les ressorts `\abovedisplayskip` et `\belowdisplayskip` en mettant l'affichage ; autrement il utilise les ressorts `\abovedisplayshortskip` et `\belowdisplayshortskip`. Voir les pages 188–189 de *The TEXbook* et 999–999 de la traduction française pour une explication plus détaillée de ce paramètre.

`\abovedisplayskip` [⟨*ressort*⟩ paramètre]
Ce paramètre spécifie le montant de ressort vertical que TEX insère avant un affichage quand l'affichage débute à gauche d'où la ligne précédente se termine, c'est-à-dire, quand il recouvre visuellement la ligne précédente. Plain TEX met `\abovedisplayskip` à `12pt plus3pt minus9pt`. Voir les pages 188–189 de *The TEXbook* et 999–999 de la traduction française pour une explication plus détaillée de ce paramètre.

`\belowdisplayskip` [⟨*ressort*⟩ paramètre]
Ce paramètre spécifie le montant de ressort vertical que TEX insère après un affichage quand l'affichage débute à gauche d'où la ligne précédente se termine, c'est-à-dire, quand il recouvre visuellement la ligne précédente. Plain TEX met `\belowdisplayskip` à `12pt plus3pt minus9pt`. Voir les pages 188–189 de *The TEXbook* et 999–999 de la traduction française pour une explication plus détaillée de ce paramètre.

`\abovedisplayshortskip` [⟨*ressort*⟩ paramètre]
Ce paramètre spécifie le montant de ressort vertical que TEX insère avant un affichage quand l'affichage débute à droite d'où la ligne précédente se termine, c'est-à-dire, quand il ne recouvre pas visuellement la ligne précédente. Plain TEX met `\abovedisplayshortskip` à `0pt plus3pt`. Voir les pages 188–189 de *The TEXbook* et 999–999 de la traduction française pour une explication plus détaillée de ce paramètre.

`\belowdisplayshortskip` [⟨*ressort*⟩ paramètre]
Ce paramètre spécifie le montant de ressort vertical que TEX insère après un affichage quand l'affichage débute à droite d'où la ligne précédente se termine, c'est-à-dire, quand il ne recouvre pas visuellement la ligne précédente. Plain TEX met `\belowdisplayshortskip` à `7pt plus3pt minus4pt`. Voir les pages 188–189 de *The TEXbook* et 999–999 de la traduction française pour une explication plus détaillée de ce paramètre.

autres paramètres d'espacement pour les mathématiques

`\mathsurround` [⟨*dimension*⟩ paramètre]
Ce paramètre spécifie le montant d'espace que TEX insère avant et après une formule mathématique en mode texte (c'est-à-dire, un formule entourée de `$` simples). Voir la page 162 de *The TEXbook* et 999 de la traduction française pour plus de détails sur son comportement. Plain TEX laisse `\mathsurround` à `0pt`.

`\nulldelimiterspace` [⟨*dimension*⟩ paramètre]
Ce paramètre spécifie la largeur de l'espace produit par un délimiteur nul. Plain TEX met `\nulldelimiterspace` à `1.2pt`.

`\scriptspace` [⟨*dimension*⟩ paramètre]
Ce paramètre spécifie le montant d'espace que TEX insère avant et après un exposant ou un indice. La commande `\nonscript` (p. 223) après un exposant ou un indice annule cet espace. Plain TEX met `\scriptspace` à `0.5pt`.

Catégoriser des constructions mathématiques

`\mathord`
`\mathop`
`\mathbin`
`\mathrel`
`\mathopen`
`\mathclose`
`\mathpunct`

Ces commandes demandent à TEX de traiter la construction qui suit comme appartenant à une classe particulière (voir la page 154 de *The TEXbook* et 999 de la traduction française pour la définition des classes). Elles sont listées ici dans l'ordre des numéros de classe, de 0 à 6. Leur effet primaire est d'ajuster l'espacement autour de la construction comme étant celui de la classe indiquée.

Exemple :
```
$\mathop{\rm minmax}\limits_{t \in A \cup B}\,t$
% By treating minmax as a math operator, we can get TeX to
% put something underneath it.
```
produit :
$\operatorname{minmax}_{t \in A \cup B}\, t$

`\mathinner`
Cette command demande à TEX de traiter la construction qui suit comme un "formule interne", c'est-à-dire, une fraction, pour l'espacement. Elle ressemble aux commandes de classe données ci-dessus.

Actions spéciales pour des formules mathématiques

`\everymath` [⟨*liste de token*⟩ paramètre]
`\everydisplay` [⟨*liste de token*⟩ paramètre]
Ces paramètres spécifient des listes de token que TEX insère au début de toute formule mathématique d'affichage ou de texte, respectivement. Vous pouvez prendre des mesures spéciales au début de chaque formule mathématique en assignant ces actions à `\everymath` ou à `\everydis-`

`play`. N'oubliez pas que si vous voulez que les deux genres de formules soient affectés, vous devez mettre les *deux* paramètres.

Exemple :

```
\everydisplay={\heartsuit\quad}
\everymath = {\clubsuit}
$3$ is greater than $2$ for large values of $3$.
$$4>3$$
```

produit :

♣3 is greater than ♣2 for large values of ♣3.

$$\heartsuit \quad 4 > 3$$

9 Commandes pour des opérations générales

Cette section couvre des caractéristiques de programmation de TEX et toutes autres choses qui ne rentre pas dans les catégories de commandes des chapitres précédents. Pour une explication des conventions utilisées dans cette section, voir "Descriptions des commandes" (p. 3).

Nommer et modifier des polices

`\font`
`\font` ⟨*séquence de contrôle*⟩ = ⟨*nom de police*⟩
`\font` ⟨*séquence de contrôle*⟩ = ⟨*nom de police*⟩ `scaled` ⟨*nombre*⟩
`\font` ⟨*séquence de contrôle*⟩ = ⟨*nom de police*⟩ `at` ⟨*dimension*⟩

Utilisée seule, la séquence de contrôle `\font` désigne la police courante. `\font` n'est pas une vraie commande quand elle est utilisée seule, puisqu'elle ne peut alors apparaître que comme argument d'une autre commande.

Pour les trois autres formes de `\font`, ⟨*nom de police*⟩ nomme un jeu de fichier qui défini une police. Ces formes de `\font` sont des commandes. Chacune de ces formes ont deux effets :

1) Elles définissent ⟨*séquence de contrôle*⟩ comme un nom qui sélectionne la police ⟨*nom de police*⟩, éventuellement magnifiée (voir plus bas).
2) Elle demande à TEX de charger le fichier de métriques de la police (fichier `.tfm`) pour ⟨*nom de police*⟩.

Le nom d'un fichier de police indique normalement sa taille de dessin. Par exemple, `cmr10` indique Computer Modern romain avec une taille de dessin de 10 points. La taille des dessins d'une police est enregistrée dans ses fichiers de métriques.

Si ni `scaled` ⟨*nombre*⟩ ni `at` ⟨*dimension*⟩ n'est présent, la police est utilisée à sa taille de dessin—la taille pour laquelle elle est normalement faite. Autrement, une version magnifiée de la police est chargée :

- Si `scaled` ⟨*nombre*⟩ est présent, la police est magnifiée d'un facteur de ⟨*nombre*⟩/1000.
- Si `at` ⟨*dimension*⟩ est présent, la police est agrandie de ⟨*dimension*⟩ en la magnifiant par ⟨*dimension*⟩/*ds*, où *ds* est la taille de dessin de ⟨*nom de police*⟩. ⟨*dimension*⟩ et *ds* sont toujours donnés en points.

Des magnifications de moins que 1 sont possible ; Elles réduisent la taille.

Vous devez habituellement fournir un fichier de formes (p. 89) pour chaque magnification de police que vous chargez. Néanmoins, certains pilotes de périphériques peuvent utiliser des polices qui sont résidentes dans une imprimante. De telles polices n'ont pas besoin de fichiers de formes.

Voir "police" (p. 88) et "magnification" (p. 76) pour plus d'information.

Exemple :

```
\font\tentt = cmtt10
\font\bigttfont = cmtt10 scaled \magstep2
\font\eleventtfont = cmtt10 at 11pt
First we use {\tentt regular CM typewriter}.
Then we use {\eleventtfont eleven-point CM typewriter}.
Finally we use {\bigttfont big CM typewriter}.
```

produit :

First we use `regular CM typewriter`. Then we use `eleven-point CM typewriter`. Finally we use `big CM typewriter`.

`\fontdimen` ⟨*nombre*⟩ ⟨*police*⟩ [⟨*dimension*⟩ paramètre]

Ce paramètre spécifie des dimensions variées associées avec la police nommée par la séquence de contrôle ⟨*font*⟩ (à distinguer de ⟨*fontname*⟩ qui nomme les fichiers de police). Des valeurs de ces paramètres sont spécifiés dans les fichiers de métriques pour ⟨*font*⟩, mais vous pouvez réparer ou changer leurs valeurs pendant une exécution de TEX. Les nombres et significations de ces paramètres sont :

Nombre	*Signification*
1	inclinaison par point
2	espace inter-mot
3	étirement inter-mot
4	rétrécissement inter-mot
5	hauteur de x (taille d'un `1ex`)
6	largeur de cadratin (taille d'un `1em`)
7	espace supplémentaire

TEX utilise l'inclinaison par point pour positionner des accents. Il utilise aussi les paramètres inter-mots pour produire des espaces inter-mots (voir `\spaceskip`, p. 113) et le paramètre d'espace supplémentaire pour l'espace additionel après une virgule (voir `\xspaceskip`, p. 113). Les valeurs

de ces paramètres pour les polices plain TEX sont énumérées dans la page 433 de *The TEXbook* et 999 de la traduction française. Les polices de symboles mathématiques ont 15 paramètres additionnel, dont nous ne parlerons pas içi.

Attention : des assignements de ces paramètres ne sont *pas* défait à la fin d'un groupe. Si vous voulez changer ces paramètres localement, vous devrez sauvegarder et restaurer leurs valeurs originales explicitement.

Exemple :

```
Here's a line printed normally.\par
\fontdimen2\font = 3\fontdimen2\font
% Triple the interword spacing.
\noindent Here's a really spaced-out line.
```

produit :

Here's a line printed normally.

Here's a really spaced-out line.

`\magnification =` ⟨*nombre*⟩
`\mag` [⟨*nombre*⟩ paramètre]

Un assigenment de `\magnification` établit le "facteur d'échelle" f qui détermine le ratio de magnification de votre document (voir "magnification", p. 76). L'assignation de `\magnification` doit apparaître avant que la première page de votre document ait été envoyée.

L'assignement met f à ⟨*nombre*⟩ et met aussi `\hsize` et `\vsize` respectively à `6.5true in` et `8.9true in`, Les valeurs approprié pour une page $8\,1/2$-par-11-pouce. f doit être entre 0 et 32768. Le ratio magnification du document est $f/1000$. Un facteur d'échelle de 1000 procure une unité de magnification, c'est-à-dire, il laisse la taille de votre document inchangée. Il est d'usage d'utiliser des puissances de 1.2 comme facteur d'échelle et la plupart des librairies de polices sont basées sur de tels facteurs. vous pouvez utiliser les commandes `\magstep` et `\magstephalf` pour spécifier des magnifications par ces facteurs.

`\magnification` n'est pas un paramètre. Vous ne pouvez pas l'utiliser pour *récuperer* le facteur d'échelle. Si vous écrivez quelque chose comme `\dimen0 = \magnification`, TEX s'en plaindra.

Le paramètre `\mag` contient le facteur d'échelle. Changer la valeur de `\mag` retaille les dimensions de la page, ce qui n'est pas ce que vous voulez habituellement. Aussi, il est préférable de changer la magnification en l'assignant à `\magnification` plutôt qu'à `\mag`.

Exemple :

```
\magnification = \magstep2
% magnify fonts by 1.44 (=1.2x1.2)
```

`\magstep` ⟨*nombre*⟩
Cette commande développe le ratio de magnification demandé pour magnifier tout dans votre document (autre que des dimensions `true`) par 1.2^r, où r est la valeur de ⟨*nombre*⟩. ⟨*nombre*⟩ doit être entre 0 et 5.

Exemple :
```
\magnification = \magstep1 % Magnify by ratio of 1.2.
```

`\magstephalf`
Cette commande développe le ratio de magnification demandé pour magnifier tout dans votre document (autre que des dimensions `true`) par $\sqrt{1.2}$, c'est-à-dire, à mi chemin entre 1 et 1.2.

Exemple :
```
\magnification = \magstephalf
```

Convertir l'information en tokens

Nombres

`\number` ⟨*nombre*⟩
Cette commande produit la représentation d'un nombre comme une séquence de tokens de caractère. Le nombre peut être un entier explicite, un paramètre ⟨*nombre*⟩ ou un registre ⟨*nombre*⟩.

Exemple :
```
\number 24 \quad \count13 = -10000 \number\count13
```
produit :
24 -10000

☞ `\romannumeral` ⟨*nombre*⟩
Cette commande produit la représentation numérique romaine d'un nombre comme une séquence de tokens de caractère. Le nombre peut être un entier explicite, un paramètre ⟨*nombre*⟩ ou un registre ⟨*nombre*⟩. Si le nombre est zéro ou négatif, `\romannumeral` ne produit pas de tokens.

Exemple :
```
\romannumeral 24 \quad (\romannumeral -16)\quad
\count13 = 6000 \romannumeral\count13
```

produit :
xxiv () mmmmmm

Information environmentale

`\time` [⟨*nombre*⟩ paramètre]
TEX met ce paramètre au nombre de minutes qui se sont déroulées depuis minuit (du jour courant). à neuf heure, par exemple, `\time` est à 720. Cette commande et les trois suivante utilisent l'heure et la date enregistrée dans votre ordinateur. TEX les récupère juste une fois, au début de votre exécution, donc `\time` à la fin de l'exécution à toujours la même valeur que `\time` au début de l'exécution (à moins que vous l'ayez changée explicitement).

`\day` [⟨*nombre*⟩ paramètre]
TEX met ce paramètre au jour courant du mois. c'est un nombre entre 1 et 31. `\day` est mis au début de votre exécution (voir les commentaires sur `\time` ci-dessus).

`\month` [⟨*nombre*⟩ paramètre]
TEX met ce paramètre au mois courant. c'est un nombre entre 1 et 12. `\month` est mis au début de votre exécution (voir les commentaires sur `\time` ci-dessus).

`\year` [⟨*nombre*⟩ paramètre]
TEX met ce paramètre à l'année courante (après J.-C.). C'est un nombre tel que 2004. `\year` est mis au début de votre exécution (voir les commentaires sur `\time` ci-dessus).

`\fmtname`
`\fmtversion`
Ces commandes produisent le nom et le numéro de version du format TEX, c'est-à-dire plain TEX or LATEX, que vous utilisez.

Exemple :
```
This book was produced with the \fmtname\ format,
version~\fmtversion.
```
produit :
This book was produced with the eplain format, version 1.9: 26 April 1991 (and plain 3.1415926).

`\jobname`
Cette commande produit le nom de base du fichier avec lequel TEX à été invoqué. Par exemple, si votre fichier d'entrée principal est `hatter.tex`,

`\jobname` se développera en `hatter`. `\jobname` est plus pratique quand vous créez un fichier auxiliaire devant être associé avec un document.

Exemple :

```
\newwrite\indexfile  \openout\indexfile = \jobname.idx
% For input file `hatter.tex', open index file `hatter.idx'.
```

Valeurs des variables

`\meaning` ⟨*token*⟩

Cette commande produit la signification de ⟨*token*⟩. Elle est pratique pour diagnostiquer en sortie. Vous pouvez utiliser la commande `\the` (p. 242) d'une manière similaire pour obtenir l'information sur les valeurs des registres et autres entités de TEX.

Exemple :

```
[{\tt \meaning\eject}] [\meaning\tenrm] [\meaning Y]
```

produit :

[macro:->\par \break] [select font cmr10] [the letter Y]

`\string` ⟨*séquence de contrôle*⟩

Cette commande produit les caractères qui forment le nom ⟨*séquence de contrôle*⟩, en incluant le caractère d'échappement. Le caractère d'échappement est représenté par la valeur courante de `\escapechar`. TEX donne les caractères dans la liste d'un code de catégorie de 12 (outre).

Vous pouvez procéder à l'opération inverse avec la commande `\csname` (p. 241), qui transforme une chaine en une séquence de contrôle.

Exemple :

```
the control sequence {\tt \string\bigbreak}
```

produit :

the control sequence \bigbreak

`\escapechar` [⟨*nombre*⟩ paramètre]

Ce paramètre spécifie le code ASCII du caractère que TEX utilise pour représenter le caractère d'échappement quand il convertit un nom de séquence de contrôle en une séquence de tokens de caractère. Cette conversion à lieu quand vous utilisez la commande `\string` et aussi quand TEX produit des messages de diagnostique. La valeur par défaut du caractère d'échappement est 92, la code de caractère ASCII pour un backslash. Si `\escapechar` n'est pas dans la fourchette 0–255, TEX n'inclue pas de caractère d'échappement dans le résultat de la conversion.

Exemple :

```
\escapechar = `!
the control sequence {\tt \string\bigbreak}
```

produit :
the control sequence `!bigbreak`

`\fontname` ⟨*police*⟩
Cette commande produit le nom de fichier pour la ⟨*police*⟩. Le nom de fichier est le ⟨*fontname*⟩ qui a été utilisé pour définir ⟨*police*⟩.

Exemple :
```
\font\myfive=cmr5 [\fontname\myfive]
```
produit :
[cmr5]

Groupement

`\begingroup`
`\endgroup`
Ces deux commandes débutent et terminent un groupe. Un `\begingroup` ne s'accode pas avec une accolade droite, ni un `\endgroup` avec une accolade gauche.

TEX traite `\begingroup` et `\endgroup` comme tout autre séquence de contrôle quand il scanne son entrée. En particulier, vous pouvez définir une macro qui contient un `\begingroup` mais pas de `\endgroup`, et inversement. Cette technique est souvent pratique quand vous définissez des paires de macros, une qui établit un environnement et l'autre qui termine cet environnement. Vous ne pouvez, de plus, pas utiliser `\begingroup` et `\endgroup` comme substituts pour des accolades autres que celle qui entourent un groupe.

Exemple :
```
\def\a{One \begingroup \it two }
\def\enda{\endgroup four}
\a three \enda
```
produit :
One *two three* four

`{`	`}`
`\bgroup`	`\egroup`

Les accolades gauches et droites sont des commandes qui débutent et terminent un groupe. Les séquences de contrôle `\bgroup` et `\egroup` sont équivalentes à '`{`' et '`}`', sauf que TEX traite `\bgroup` et `\egroup` comme tout autre séquence de contrôle quand il scanne son entrée.

`\bgroup` et `\egroup` peuvent être pratique quand vous définissez des paires de macros, une dans laquelle débute une construction délimitée par accolade (pas nécessairement un groupe) et l'autre dans laquelle ce

termine cette construction. Vous ne pouvez pas définir de telles macros en utilisant des accolades ordinaire—si vous essayez, vos définitions de macro contiendra des accolades non appairées et sera de ce fait inacceptable par TEX. Normalement vous ne devez utilisez ces commandes que quand vous ne pouvez utiliser d'accolades ordinaires.

Exemple :

```
Braces define the {\it boundaries\/} of a group.
```

produit :

Braces define the *boundaries* of a group.

Exemple :

```
\def\a{One \vbox\bgroup}
% You couldn't use { instead of \bgroup  here because
% TeX would not recognize the end of the macro
\def\enda#1{{#1\egroup} two}
% This one is a little tricky, since the \egroup actually
% matches a left brace and the following right brace
% matches the \bgroup.  But it works!
\a \enda{\hrule width 1in}
```

produit :

One _____ two

`\global`

Cette commande rend la définition ou l'assignement suivant global (voir "global", p. 70) pour qu'il devienne effectif indépendamment des frontières du groupe. Vous pouvez appliquer un préfix `\global` à toute sorte de définition ou assignement, en incluant une définition de macro ou un assignement de registre.

Exemple :

```
{\global\let\la = \leftarrow}
$a \la b$
```

produit :

$a \leftarrow b$

`\globaldefs` [⟨*nombre*⟩ paramètre]

Ce paramètre contrôle si TEX prend ou non des définitions et autres assignements comme étant globaux :

- Si `\globaldefs` est à zéro (comme il l'est par défaut), une définition est globale si et seulement si il est précédé par `\global` soit explicitement soit implicitement. (Les commandes `\gdef` et `\xdef` (p. 239) ont un préfixe `\global` implicite).
- Si `\globaldefs` est supérieur zéro, tout assignements et définitions sont implicitement préfixé par `\global`.
- Si `\globaldefs` est inférieur à zéro, tout préfixe `\global` est ignoré.

`\aftergroup` ⟨*token*⟩
Quand TEX rencontre cette commande dans une entrée, il sauve ⟨*token*⟩. Après la fin du groupe courant, il réinsère ⟨*token*⟩ dans l'entrée et le développe. Si un groupe contient plusieurs `\aftergroup`s, les tokens correspondants sont *tous* insérés suivant la fin du groupe, dans l'ordre dans lequel ils apparaissent à l'origine.

L'exemple qui suit vous montre comment vous pouvez utiliser `\aftergroup` pour différer le traitement d'un token que vous avez généré dans un test conditionnel.

Exemple :
```
\def\neg{negative} \def\pos{positive}
% These definitions are needed because \aftergroup applies
% to a single token, not to a sequence of tokens or even
% to a brace-delimited text.
\def\arith#1{Is $#1>0$? \begingroup
   \ifnum #1>-1 Yes\aftergroup\pos
   \else No\aftergroup\neg\fi
   , it's \endgroup. }
\arith 2
\arith {-1}
```
produit :
Is $2 > 0$? Yes, it's positive. Is $-1 > 0$? No, it's negative.

`\afterassignment` ⟨*token*⟩
Quand TEX rencontre cette commande il sauve ⟨*token*⟩ dans un endroit spécial. Après qu'il ait exécuté un assignment suivant, il insère ⟨*token*⟩ dans l'entrée et le développe. Si vous appelez `\afterassignment` plus d'une fois avant un assignement, seul le dernier appel a un effet. Une utilisation de `\afterassignment` est dans l'écriture de macros pour des commandes devant être écrite sous la forme d'assignements, comme dans l'exemple suivant.

Voir la page 279 de *The TEXbook* et 999 de la traduction française pour un description précise du fonctionnement d'`\afterassignment`.

Exemple :
```
\def\setme{\afterassignment\setmeA\count255}
\def\setmeA{$\number\count255\advance\count255 by 10
   +10=\number\count255$}
Some arithmetic: \setme = 27
% After expanding \setme, TeX sets \count255 to 27 and
% then calls \setmeA.
```
produit :
Some arithmetic: $27 + 10 = 37$

Macros

Définir des macros

`\def` ⟨*séquence de contrôle*⟩ ⟨*texte de paramètre*⟩ `{` ⟨*texte de remplacement*⟩ `}`

Cette commande définit ⟨*séquence de contrôle*⟩ comme une macro avec les ⟨*texte de paramètre*⟩ et ⟨*texte de remplacement*⟩ spécifiés. Voir page 73 pour une explication complète sur la façon d'écrire une definition de macro.

Exemple :

```
\def\add#1+#2=?{#1+#2&=
   \count255=#1 \advance\count255 by #2 \number\count255\cr}
$$\eqalign{
   \add 27+9=?
   \add -5+-8=?}$$
```

produit :

$$27 + 9 = 36$$
$$-5 + -8 = -13$$

`\edef` ⟨*séquence de contrôle*⟩ ⟨*texte de paramètre*⟩ `{` ⟨*texte de remplacement*⟩ `}`

Cette commande définit une macro de la même manière générale que `\def`. La différence est que TEX développe le ⟨*texte de remplacement*⟩ d'un `\edef` immédiatement (mais sans rien exécuter). Ainsi toutes définitions du ⟨*texte de remplacement*⟩ sont développées, mais des assignements et commandes qui produisent des choses telles que boîtes et des ressorts sont laissé tels quels. Par exemple, une commande `\hbox` dans le ⟨*texte de remplacement*⟩ d'un `\edef` reste comme une commande et n'est pas transformée en une boîte quand TEX exécute la définition. Il n'est pas toujours évident de savoir ce qui est développé et ce qui ne l'est pas, mais vous trouverez une liste complète des séquences de contrôle développable sur les pages 212–215 de *The TEXbook* et 999–999 de la traduction française.

Vous pouvez inhiber le développement d'une séquence de contrôle qui devrait être développé en utilisant `\noexpand` (p. 242). Vous pouvez différer le développement d'une séquence de contrôle en utilisant la commande `\expandafter` (p. 241).

Les commandes `\write`, `\message`, `\errmessage`, `\wlog` et `\csname` développent leurs liste de token en utilisant les même règles que `\edef` utilise pour développer son texte de remplacement.

Exemple :

```
\def\aa{xy} \count255 = 1
\edef\bb{w\ifnum \count255 > 0\aa\fi z}
% equivalent to \def\bb{wxyz}
\def\aa{} \count255 = 0 % leaves \bb unaffected
\bb
```

produit :

wxyz

`\gdef` ⟨*séquence de contrôle*⟩ ⟨*texte de paramètre*⟩ `{` ⟨*texte de remplacement*⟩ `}`

Cette commande est équivalente à `\global\def`.

`\xdef` ⟨*séquence de contrôle*⟩ ⟨*texte de paramètre*⟩ `{` ⟨*texte de remplacement*⟩ `}`

Cette commande est équivalente à `\global\edef`.

`\long`

Cette commande est utilisée comme préfixe d'une définition de macro. Elle dis à TEX que les arguments de la macro peuvent inclure des tokens `\par` (p. 116), qui indique normalement la fin d'un paragraphe. Si TEX essaye de développer une macro définie sans `\long` et qu'un des arguments de la macro comprend un token `\par`, TEX se plaindra par un "runaway argument". Le but de ce comportement est de vous fournir des protections contre des arguments de macro non terminés. `\long` vous donne un moyen d'outrepasser la protection.

Exemple :

```
\long\def\aa#1{\par\hrule\smallskip#1\par\smallskip\hrule}
\aa{This is the first line.\par
This is the second line.}
% without \long, TeX would complain
```

produit :

This is the first line.
This is the second line.

`\outer`

Cette commande est utilisée comme préfixe d'une définition de macro. Elle dis à TEX que la macro est externe (p. 84) et ne peut pas être utilisée

dans certains contextes. Si la macro est utilisée dans un contexte interdit, TEX se plaindra.

Exemple :

```
\outer\def\chapterhead#1{%
   \eject\topglue 2in \centerline{\bf #1}\bigskip}
% Using \chapterhead in a forbidden context causes an
% error message.
```

`\chardef` ⟨*séquence de contrôle*⟩=⟨*charcode*⟩
Cette commande définit ⟨*séquence de contrôle*⟩ comme étant ⟨*charcode*⟩. Bien que `\chardef` soit plus souvent utilisée pour définir des caractères, vous pouvez aussi l'utiliser pour donner un nom à un nombre dans la fourchette 0–255 même quand vous n'utilisez pas ce nombre comme un code de caractère.

Exemple :

```
\chardef\percent = `\% 21\percent, {\it 19\percent}
% Get the percent character in roman and in italic
```

produit :

21%, *19%*

`\mathchardef` ⟨*séquence de contrôle*⟩=⟨*mathcode*⟩
Cette commande définit la ⟨*séquence de contrôle*⟩ comme un caractère mathématique avec le ⟨*mathcode*⟩ donné. La séquence de contrôle ne sera légale qu'en mode mathématique.

Exemple :

```
\mathchardef\alphachar = "010B % As in plain TeX.
$\alphachar$
```

produit :

α

Autre définitions

`\let` ⟨*séquence de contrôle*⟩ = ⟨*token*⟩
Cette commande fait que ⟨*séquence de contrôle*⟩ prend la signification actuelle de ⟨*token*⟩. Même si vous redéfinissez ⟨*token*⟩ plus tard, la signification de ⟨*séquence de contrôle*⟩ ne changera pas. Bien que ⟨*token*⟩ soit le plus souvent une séquence de contrôle, il peut aussi être un token de caractère.

`\futurelet` ⟨*séquence de contrôle*⟩ ⟨$token_1$⟩ ⟨$token_2$⟩
Cette commande dis à TEX de donner à ⟨$token_2$⟩ la signification de ⟨*séquence de contrôle*⟩ (comme elle le serait avec `\let`) et ensuite d'exécuter ⟨$token_1$⟩ et ⟨$token_2$⟩ normalement. `\futurelet` est pratique à la fin

d'une définition de macro parce qu'elle vous donne un moyen de regarder au-delà du token que TEX est en train d'exécuter avant de continuer.

Exemple :

```
\def\predict#1{\toks0={#1}\futurelet\next\printer}
% \next will acquire the punctuation mark after the
% argument to \predict
\def\printer#1{A \punc\ lies ahead for \the\toks0. }
\def\punc{%
   \ifx\next;semicolon\else
      \ifx\next,comma\else
         ``\next''\fi\fi}
\predict{March}; \predict{April}, \predict{July}/
```

produit :

A semicolon lies ahead for March. A comma lies ahead for April. A "/" lies ahead for July.

`\csname` ⟨*liste de token*⟩ `\endcsname`

Cette commande produit une séquence de contrôle à partir de ⟨*liste de token*⟩. Elle procure un moyen de synthétiser des séquence de contrôles, en incluant celle que vous ne pouvez normalement pas écrire. ⟨*liste de token*⟩ peut elle-même inclure des séquences de contrôles ; elle est développable de la même manière que le texte de remplacement d'une définition `\edef` (p. 238). Si le développement final produit quelque chose qui n'est pas un caractère, TEX se plaindra. `\csname` transforme une liste de tokens en séquence de contrôle ; vous pouvez faire l'inverse avec `\string` (p. 234).

Exemple :

```
\def\capTe{Te}
This book purports to be about \csname\capTe X\endcsname.
```

produit :

This book purports to be about TEX.

Contrôler le développement

`\expandafter` ⟨$token_1$⟩ ⟨$token_2$⟩

Cette commande dit à TEX de développer ⟨$token_1$⟩ en accord avec ses règles de développement de macro *après* avoir développé ⟨$token_2$⟩ d'un niveau. C'est pratique quand ⟨$token_1$⟩ est quelque chose comme '`{`' ou `\string` qui inhibe les expansions de ⟨$token_2$⟩, mais que vous voulez développer ⟨$token_2$⟩ quand même.

Exemple :

```
\def\aa{xyz}
\tt % Use this font so '\' prints that way.
[\string\aa]  [\expandafter\string\aa]
[\expandafter\string\csname TeX\endcsname]
```

produit :

```
[\aa] [xyz] [\TeX]
```

`\noexpand` ⟨*token*⟩

Cette commande demande à TEX de supprimer le développement de ⟨*token*⟩ si ⟨*token*⟩ est une séquence de contrôle qui peut être développée. Si ⟨*token*⟩ ne peut pas être développée, c'est-à-dire, si c'est une lettre, TEX agit comme si le `\noexpand` n'était pas là et exécute ⟨*token*⟩ normalement. En d'autres mots le développement de '`\noexpand`⟨*token*⟩' est simplement ⟨*token*⟩ quoi que ⟨*token*⟩ devienne.

Exemple :

```
\def\bunny{rabbit}
\edef\magic{Pull the \noexpand\bunny\ out of the hat! }
% Without \noexpand, \bunny would always be replaced
% by 'rabbit'
\let\oldbunny=\bunny \def\bunny{lagomorph} \magic
\let\bunny=\oldbunny \magic
```

produit :

Pull the lagomorph out of the hat! Pull the rabbit out of the hat!

`\the` ⟨*token*⟩

Cette commande se développe généralement en une liste de tokens de caractère que représente ⟨*token*⟩. ⟨*token*⟩ peut être parmi ce qui suit :

- un paramètre TEX, par exemple, `\parindent` ou `\deadcycles`
- un registre, par exemple, `\count0`
- un code associé avec un caractère entré, par exemple, `\catcode`(`
- un paramètre de police, par exemple, `\fontdimen3\sevenbf`
- le `\hyphenchar` ou le `\skewchar` d'une police, `\skewchar\teni`, par exemple
- `\lastpenalty`, `\lastskip` ou `\lastkern` (valeurs dérivée du dernier élément d'une liste horizontale ou liste verticale courante)
- une séquence de contrôle définie par `\chardef` ou `\mathchardef`

De plus, `\the` peut développer des tokens non-caractère dans les deux cas suivants :

- `\the` ⟨*police*⟩, qui développe la séquence de contrôle la plus récemment définie qui sélectionne la même police que la séquence de contrôle ⟨*police*⟩
- `\the` ⟨*variable de token*⟩, qui développe une copie de la valeur de la variable, par exemple, `\the\everypar`

Voir les pages 214–215 de *The TEXbook* et 999–999 de la traduction française pour une description plus détaillée de ce que `\the` fait selon les cas.

Exemple :

```
The vertical size is currently \the\vsize.
The category code of `(' is \the\catcode `(.
```

produit :

The vertical size is currently 573.96269pt. The category code of '(' is 12.

Voir aussi : "Convertir l'information en tokens" (p. 232) ainsi que la commande `\showthe` (p. 261).

Tests conditionnels

`\if` ⟨*token*$_1$⟩ ⟨*token*$_2$⟩

Cette commande teste si ⟨*token*$_1$⟩ et ⟨*token*$_2$⟩ ont le même code de caractère, indépendamment de leurs codes de catégorie. Avant d'exécuter le test, TEX développe les tokens suivant le `\if` jusqu'a ce qu'il obtienne deux tokens qui ne peuvent se développer plus. Ces deux tokens deviennent ⟨*token*$_1$⟩ et ⟨*token*$_2$⟩. Le développement inclut le remplacement d'un séquence de contrôle `\let` égale à un token de caractère par ce token de caractère. Une séquence de contrôle qui ne peut plus se développer plus est considérée comme ayant un code de caractère 256.

Exemple :

```
\def\first{abc}
\if\first true\else false\fi;
% ``c'' is left over from the expansion of \first.
% It lands in the unexecuted ``true'' part.
\if a\first\ true\else false\fi;
% Here ``bc'' is left over from the expansion of \first
\if \hbox\relax true\else false\fi
% Unexpandable control sequences test equal with ``if''
```

produit :

false; bc true; true

`\ifcat` ⟨*token*$_1$⟩ ⟨*token*$_2$⟩

Cette commande teste si ⟨*token*$_1$⟩ et ⟨*token*$_2$⟩ ont le même code de catégorie. Avant d'exécuter le test, TEX développe les tokens suivant le `\if` jusqu'a ce qu'il obtienne deux tokens qui ne peuvent se développer plus. Ces deux tokens deviennent ⟨*token*$_1$⟩ et ⟨*token*$_2$⟩. Le développement inclut le remplacement d'un séquence de contrôle `\let` égale à un token de caractère par ce token de caractère. Une séquence de contrôle qui

ne peut plus se développer plus est considérée comme ayant un code de caractère 16.

Exemple :

```
\ifcat axtrue\else false\fi;
\ifcat ]}true\else false\fi;
\ifcat \hbox\day true\else false\fi;
\def\first{12345}
\ifcat (\first true\else false\fi
% ``2345'' lands in the true branch of the test
```

produit :

true; false; true; 2345true

`\ifx` ⟨*token*$_1$⟩ ⟨*token*$_2$⟩

Cette commande teste si ⟨*token*$_1$⟩ et ⟨*token*$_2$⟩ s'accordent. Contrairement à `\if` et `\ifcat`, `\ifx` ne développent *pas* les tokens suivant `\ifx`, donc ⟨*token*$_1$⟩ et ⟨*token*$_2$⟩ sont les deux tokens immédiatement après `\ifx`. Il y a trois cas :

1) Si un token est une macro et pas l'autre, les tokens ne s'accordent pas.
2) Si aucun token n'est une macro, Les tokens s'accordent si :
   a) les deux tokens sont des caractères (ou des séquences de contrôle décrivant des caractères) et que leurs codes de caractère et de catégorie s'accordent, ou
   b) Les deux tokens font référence à la même commande TEX, police, etc.
3) Si les deux tokens sont des macros, les tokens s'accordent si :
   a) leurs développements de "premier niveau", c'est-à-dire, leurs textes de remplacement, sont identique, et
   b) Ils ont le même statut concernant `\long` (p. 239) et `\outer` (p. 239).

   Notez en particulier que *deux séquences de contrôle non définies s'accordent.*

Ce teste est généralement plus utile que `\if`.

Exemple :

```
\ifx\alice\rabbit true\else false\fi;
% true since neither \rabbit nor \alice is defined
\def\a{a}%
\ifx a\a true\else false\fi;
% false since one token is a macro and the other isn't
\def\first{\a}\def\second{\aa}\def\aa{a}%
\ifx \first\second true\else false\fi;
% false since top level expansions aren't the same
\def\third#1:{(#1)}\def\fourth#1?{(#1)}%
\ifx\third\fourth true\else false\fi
% false since parameter texts differ
```

produit :

true; false; false; false

`\ifnum` ⟨*number*$_1$⟩ ⟨*relation*⟩ ⟨*number*$_2$⟩

Cette commande teste si ⟨*number*$_1$⟩ et ⟨*number*$_2$⟩ satisfont ⟨*relation*⟩, qui peut être soit '`<`', '`=`' ou '`>`'. Les nombres peuvent être des constantes telles que `127`, des registres de compteur tels que `\pageno` ou `\count22` ou des paramètres numérique tels que `\hbadness`. Avant d'exécuter le test, TEX développe les tokens suivant le `\ifnum` jusqu'a ce qu'il obtienne une séquence de tokens ayant la forme ⟨*number*$_1$⟩ ⟨*relation*⟩ ⟨*number*$_2$⟩ suivi par un token qui ne peut être une partie de ⟨*number*$_2$⟩.

Exemple :

```
\count255 = 19 \ifnum \count255 > 12 true\else false\fi
```

produit :

true

`\ifodd` ⟨*nombre*⟩

Cette commande teste si ⟨*nombre*⟩ est impair. Avant d'exécuter le test, TEX développe les tokens suivant le `\ifodd` jusqu'a ce qu'il obtienne une séquence de tokens ayant la forme ⟨*nombre*⟩, suivi par un token qui ne peut être une partie de ⟨*nombre*⟩.

Exemple :

```
\count255 = 19
\ifodd 5 true\else false\fi
```

produit :

true

`\ifdim` ⟨*dimen*$_1$⟩ ⟨*relation*⟩ ⟨*dimen*$_2$⟩

Cette commande teste si ⟨*dimen*$_1$⟩ et ⟨*dimen*$_2$⟩ satisfont ⟨*relation*⟩, qui doit être soit '`<`', '`=`' ou '`>`'. Les dimensions peuvent être des constantes telles que `1in`, des registres de dimension tels que `\dimen6` ou des paramètres de dimension tels que `\parindent`. Avant d'exécuter le test,

TEX développe les tokens suivant le `\ifdim` jusqu'a ce qu'il obtienne une séquence de tokens ayant la forme ⟨*dimen*$_1$⟩ ⟨*relation*⟩ ⟨*dimen*$_2$⟩ suivi par un token qui ne peut être une partie de ⟨*dimen*$_2$⟩.

Exemple :

```
\dimen0 = 1000pt \ifdim \dimen0 > 3in true\else false\fi
```

produit :

true

`\ifhmode`
`\ifvmode`
`\ifmmode`
`\ifinner`

Ces commandes testent dans quel mode est TEX :

- `\ifhmode` est vrai si TEX est en mode horizontal ordinaire ou réduit.
- `\ifvmode` est vrai si TEX est en mode vertical ordinaire ou interne.
- `\ifmmode` est vrai si TEX est en mode mathématique de texte ou d'affichage.
- `\ifinner` est vrai si TEX est dans un mode "interne" : horizontal réduit, vertical interne ou mathématique de texte.

Exemple :

```
\def\modes{{\bf
   \ifhmode
      \ifinner IH\else H\fi
   \else\ifvmode
      \ifinner \hbox{IV}\else \hbox{V}\fi
   \else\ifmmode \hbox{M}\else
      error\fi\fi\fi}}
Formula $\modes$; then \modes,
   \hbox{next \modes\ and \vbox{\modes}}.
\par\modes
```

produit :

Formula **M**; then **H**, next **IH** and **IV**.
V

`\ifhbox` ⟨*registre*⟩
`\ifvbox` ⟨*registre*⟩
`\ifvoid` ⟨*registre*⟩

Ces commandes testent le contenu du registre de boîte numéro ⟨*register*⟩. Soit ⟨*registre*⟩ à n. Alors :

- `\ifhbox` est vrai si `\box`n est une hbox.
- `\ifvbox` est vrai si `\box`n est une vbox.
- `\ifvoid` est vrai si `\box`n est vide, c'est-à-dire, ne contient pas de boîte.

Exemple :

```
\setbox0 = \vbox{} % empty but not void
\setbox1 = \hbox{a}
\setbox2 = \box1 % makes box1 void
\ifvbox0 true\else false\fi;
\ifhbox2 true\else false\fi;
\ifvoid1 true\else false\fi
```

produit :

true; true; true

`\ifeof` ⟨*nombre*⟩

Cette commande teste un flux d'entrée de fin de fichier. Elle est vraie si le flux d'entrée ⟨*nombre*⟩ n'a pas été ouvert, ou a été ouvert et que le fichier associé a été entièrement lu (ou n'existe pas).

`\ifcase` ⟨*nombre*⟩⟨*texte du cas*$_0$⟩ `\or` ⟨*texte du cas*$_1$⟩ `\or` ... `\or` ⟨*texte du cas*$_n$⟩
`\else` ⟨*texte alternatif*⟩ `\fi`

Cette commande introduit un test avec des cas multiples numérotés. Si ⟨*nombre*⟩ a la valeur k, TEX développera ⟨*texte de cas*$_k$⟩ s'il existe et ⟨*texte alternatif*⟩ autrement. Vous pouvez omettre le `\else`—dans ce cas, TEX ne développera rien si aucun des cas n'est satisfait.

Exemple :

```
\def\whichday#1{\ifcase #1<day 0>\or Sunday\or Monday%
   \or Tuesday\or Wednesday\or Thursday\or Friday%
   \or Saturday\else Nonday\fi
   \ is day \##1. }
\whichday2 \whichday3 \whichday9
```

produit :

Monday is day #2. Tuesday is day #3. Nonday is day #9.

`\iftrue`
`\iffalse`

Ces commandes sont équivalente à des tests qui sont toujours vrai ou toujours faux. Le principal usage de ces commandes est de définir des macros qui gardent trace du résultat d'un test.

Exemple :
```
\def\isbigger{\let\bigger=\iftrue}
\def\isnotbigger{\let\bigger=\iffalse}
% These \let's MUST be buried in macros!  If they aren't,
% TeX erroneously tries to match them with \fi.
\def\test#1#2{\ifnum #1>#2 \isbigger\else\isnotbigger\fi}
\test{3}{6}
\bigger$3>6$\else$3\le6$\fi
```
produit :

$3 \leq 6$

`\else`

Cette commande introduit l'alternative "faux" d'un test conditionel.

`\fi`

Cette commande termine le texte d'un test conditionel.

`\newif \if`⟨*nom de test*⟩

Cette commande nommes un trio de séquence de contrôles avec comme noms `\alphatrue`, `\alphafalse`, et `\ifalpha`, où `alpha` est ⟨*test name*⟩. Vous pouvez les utiliser pour définir vos propre tests en créant une variable logique qui enregistre des informations true/false:

- `\alphatrue` met la variable logique `alpha` à vrai.
- `\alphafalse` met la variable logique `alpha` à faux
- `\ifalpha` est un test conditionnel qui est vrai si la variable logique `alpha` est vraie et faux autrement.

La variable logique `alpha` n'existe pas réellement, mais TEX agit comme si elle l'était. Après `\newif\ifalpha`, la variable logique est initialisée à faux.

`\newif` est un commande externe, donc vous ne pouvez l'utiliser dans une définition de macro.

Exemple :
```
\newif\iflong  \longtrue
\iflong Rabbits have long ears.
\else Rabbits don't have long ears.\fi
```
produit :

Rabbits have long ears.

Actions répétées

`\loop` α `\if`Ω β `\repeat`
`\repeat`

Ces commandes procurent une construction de boucle pour TEX. Ici, α et β sont des séquences de commandes arbitraires et `\if`Ω est un des tests

conditionnels décrites dans "Tests conditionnels" (p. 243). Le `\repeat` répète le `\fi` correspondant au test, donc vous n'aurez pas besoin d'écrire un `\fi` explicite pour terminer le test. Ni, malheureusement, associer un `\else` au test. Si vous voulez utiliser le test dans le sens opposé, vous devez réarranger le test ou définir un test auxiliaire avec `\newif` (voir au-dessus) et utiliser ce test dans le sens que vous voulez (voir le second exemple ci-dessous).

TEX développe `\loop` comme suit :

1) α est développé.
2) `\if`Ω est exécuté. Si le résultat est faux, la boucle est terminée.
3) β est développé.
4) Le cycle est répété.

Exemple :

```
\count255 = 6
\loop
   \number\count255\
   \ifnum\count255 > 0
      \advance\count255 by -1
\repeat
```

produit :

6 5 4 3 2 1 0

Exemple :

```
\newif\ifnotdone % \newif uses \count255 in its definition
\count255=6
\loop
   \number\count255\
   \ifnum\count255 < 1 \notdonefalse\else\notdonetrue\fi
   \ifnotdone
      \advance\count255 by -1
\repeat
```

produit :

6 5 4 3 2 1 0

Ne rien faire

`\relax`

Cette commande demande à TEX de ne rien faire. C'est utile dans un contexte où vous devez procurer une commande mais il n'y a rien que vous voulez que TEX fasse.

Exemple :

```
\def\medspace{\hskip 12pt\relax}
% The \relax guards against the possibility that
% The next tokens are 'plus' or 'minus'.
```

`\empty`
Cette commande se développe en aucun tokens du tout. Elle diffère de `\relax` car elle disparaît après un développement de macro.

Registres

Utiliser des registres

`\count` ⟨*registre*⟩ = ⟨*nombre*⟩	`\count` ⟨*registre*⟩
`\dimen` ⟨*registre*⟩ = ⟨*dimen*⟩	`\dimen` ⟨*registre*⟩
`\skip` ⟨*registre*⟩ = ⟨*glue*⟩	`\skip` ⟨*registre*⟩
`\muskip` ⟨*registre*⟩ = ⟨*muglue*⟩	`\muskip` ⟨*registre*⟩
`\toks` ⟨*registre*⟩ = ⟨*token variable*⟩	`\toks` ⟨*registre*⟩
`\toks` ⟨*registre*⟩ = `{` ⟨*list. de tok.*⟩ `}`	

Les six premières commandes listées ici assigne quelque chose à un registre. Les = dans les assignations sont optionnels. Les cinq séquences de contrôle restantes ne sont pas de vraies commandes car elles n'apparaissent que comme partie d'un argument. Elles précisent le contenu des registres spécifiés. Bien que vous ne puissiez utiliser ces séquences de contrôles elles-même comme commandes dans un texte, vous pouvez utiliser `\the` pour les convertir en texte et ainsi pouvoir composer leurs valeurs.

Vous pouvez nommer et réserver de nouveaux registres avec la commande `\newcount` et ses relatifs (p. 252). Utiliser ces commandes est un moyen sûr d'obtenir des registres qui ne sont pas cencé avoir d'usages conflictuels.

Un registre `\count` contient un entier, qui peut être soit positif soit négatif. Des entiers peuvent être aussi grand que vous n'aurez probablement jamais besoin qu'ils soient.[1] TEX utilise les registres compteur 0–9 pour garder trace des numéro de page (voir la page 119 de *The TEXbook* et 999 de la traduction française). `\count255` est le seul registre compteur possible pour une utilisation sans réservation.

Exemple :

```
\count255 = 17 \number\count255
```

produit :

17

[1] Voici le seul exercice du livre : trouvez le plus grand entier que TEX acceptera.

Un registre `\dimen` contient une dimension. Les registres de `\dimen0` à `\dimen255` sont possible pour un brouillon.

Exemple :

```
\dimen0 = 2.5in
\hbox to \dimen0{$\Leftarrow$\hfil$\Rightarrow$}
```

produit :

Un registre `\skip` contient les dimensions de ressort. Différent d'un registre `\dimen`, il enregistre un montant d'étirement et de rétrécissement autant qu'une taille naturelle. Les registres de `\skip0` à `\skip9` et `\skip255` peuvent être utiliser sans réservation.

Exemple :

```
\skip2 = 2in
$\Rightarrow$\hskip \skip2 $\Leftarrow$
```

produit :

Un registre `\muskip` est comme un registre `\skip`, mais le ressort qu'il contient est toujours mesurée en `mu` (voir "unité mathématique", p. 100). La taille d'un `mu` dépend de la police courante. Par exemple, elle est normalement un peu plus petite dans un exposant que dans un texte ordinaire. Les registres de `\muskip0` à `\muskip9` et `\muskip255` peuvent être utilisé sans réservation.

Exemple :

```
\muskip0 = 24mu % An em and a half, no stretch or shrink.
$\mathop{a \mskip\muskip0 b}\limits^{a \mskip\muskip0 b}$
% Note the difference in spacing.
```

produit :

$\mathop{a \mskip\muskip0 b}\limits^{a \mskip\muskip0 b}$

Vous pouvez assigner soit un token variable (un registre ou un paramètre) ou une liste de token à un registre `\toks`. Quand vous assignez une liste de token à un registre de token, les tokens de la liste de token ne sont *pas* développé.

Une fois que les tokens d'une liste de token ont été inséré dans un texte en utilisant `\the`, ils sont développé comme des tokens qui seraient lu directement. Ils ont les codes de catégorie qu'il ont reçu quand TEX les à vu une première fois dans la source.

Exemple :

```
\toks0 = {the \oystereaters\ were at the seashore}
% This assignment doesn't expand \oystereaters.
\def\oystereaters{Walrus and Carpenter}
\toks1 = \toks0
% the same tokens are now in \toks0 and \toks1
Alice inquired as to whether \the\toks1.
```

produit :

Alice inquired as to whether the Walrus and Carpenter were at the seashore.

`\maxdimen`

Cette séquence de contrôle désigne une ⟨*dimen*⟩ qui est la plus grande dimension acceptable par TEX (environ 4,50 mètres). Ce n'est pas une vraie commande parce qu'elle ne peut apparaître que comme partie d'un argument d'une autre commande.

Exemple :

```
\maxdepth = \maxdimen % Remove restrictions on \maxdepth.
```

Voir aussi : `\advance` (p. 253), `\multiply`, `\divide` (p. 254), `\setbox`, `\box` (p. 170).

Nommer et réserver des registres, etc.

`\newcount` `\newdimen` `\newskip` `\newmuskip` `\newtoks` `\newbox` `\newread` `\newwrite` `\newfam` `\newinsert` `\newlanguage`

Ces commandes réservent et nomment une entité du type indiqué :

- `\newcount`, `\newdimen`, `\newskip`, `\newmuskip`, `\newtoks`, et `\newbox` réservent chacun un registre du type indiqué.
- `\newread` et `\newwrite` réservent un flot d'entrée et de sortie respectivement.
- `\newfam` réserve une famille de police mathématique.
- `\newinsert` réserve un caractère d'insertion. (Réserver un caractère d'insertion entraîne réserver plusieurs registres différents.)
- `\newlanguage` réserve un jeu de motifs de césure.

Vous devez utiliser ces commandes à chaque fois que vous avez besoin d'une de ces entités, à part dans une région très locale, pour éviter des conflits de numérotation.

Il y a une différence importante entre ces commandes :

- Les séquences de contrôle définies par `\newcount`, `\newdimen`, `\newskip`, `\newmuskip` et `\newtoks` désignent chacune une entité du type approprié. Par exemple, après la commande :

  ```
  \newdimen\listdimen
  ```

  la séquence de contrôle `\listdimen` peut être utilisée comme une dimension.
- Les séquences de contrôle définies par `\newbox`, `\newread`, `\newwrite`, `\newfam`, `\newinsert` et `\newlanguage` évaluent chacune le *numéro* d'une entité du type approprié. Par exemple, après la commande :

  ```
  \newbox\figbox
  ```

  La séquence de contrôle `\figbox` doit être utilisée en conjonction avec une commande de type `\box`, par exemple :

  ```
  \setbox\figbox = \vbox{...}
  ```

`\countdef` ⟨*séquence de contrôle*⟩ `=` ⟨*registre*⟩
`\dimendef` ⟨*séquence de contrôle*⟩ `=` ⟨*registre*⟩
`\skipdef` ⟨*séquence de contrôle*⟩ `=` ⟨*registre*⟩
`\muskipdef` ⟨*séquence de contrôle*⟩ `=` ⟨*registre*⟩
`\toksdef` ⟨*séquence de contrôle*⟩ `=` ⟨*registre*⟩
Ces commandes définissent ⟨*séquence de contrôle*⟩ pour faire référence au registre de la catégorie indiquée dont le numéro est ⟨*registre*⟩. Normalement vous devez utiliser les commandes du groupe précédent (`\newcount`, etc.) en préférence à ces commandes pour éviter des conflits de numéro. Les commandes du groupe précédent sont définies en termes de commandes de ce groupe.

Exemple :

```
\countdef\hatters = 19 % \hatters now refers to \count19
\toksdef\hares = 200 % \hares now refers to \toks200
```

Voir aussi : `\newif` (p. 248), `\newhelp` (p. 270).

Faire de l'arithmétique dans des registres

`\advance` ⟨*registre compteur*⟩ `by` ⟨*nombre*⟩
`\advance` ⟨*registre dimension*⟩ `by` ⟨*dimension*⟩
`\advance` ⟨*registre saut*⟩ `by` ⟨*ressort*⟩
`\advance` ⟨*registre muskip*⟩ `by` ⟨*muglue*⟩
Cette commande ajoute une quantité compatible a un registre. Pour ⟨*ressort*⟩ ou ⟨*muglue*⟩ les trois composants (valeur naturelle, étirement et rétrécissement) sont ajouté. toutes les quantités peuvent être négatives.

Pour ces calculs (et autres assignements en fait), ⟨*ressort*⟩ peut être converti en une ⟨*dimension*⟩ en enlevant l'étirement et le rétrécissement et une ⟨*dimension*⟩ peut être convertie en ⟨*nombre*⟩ en prenant sa valeur en points d'échelle (voir "dimension", p. 61). Vous pouvez omettre le mot `by` dans ces commandes—TEX les comprendra quand même.

Exemple :

```
\count0 = 18 \advance\count0 by -1 \number\count0\par
\skip0 = .5in \advance\skip0 by 0in plus 1in % add stretch
\hbox to 2in{a\hskip\skip0 b}
```

produit :

17

a b

`\multiply` ⟨*registre*⟩ `by` ⟨*nombre*⟩
`\divide` ⟨*registre*⟩ `by` ⟨*nombre*⟩

Ces commandes multiplient et divisent la valeur de ⟨*registre*⟩ par ⟨*nombre*⟩ (qui peut être négatif). Le registre peut être un registre `\count`, `\dimen`, `\skip` ou `\muskip`. Pour un registre `\skip` ou `\muskip` (p. 250), les trois composants du ressort du registre sont modifiés. Vous pouvez omettre le mot `by` dans ces commandes—TEX les comprendra quand même.

Vous pouvez aussi obtenir un multiple d'une ⟨*dimension*⟩ en la précédant par un ⟨*nombre*⟩ ou une constante décimale, par exemple, `-2.5\dimen2`. Vous pouvez aussi utiliser cette notation pour des ⟨*ressort*⟩, mais attention—le résultat est une ⟨*dimension*⟩, pas un ⟨*ressort*⟩. Ainsi `2\baselineskip` donne une ⟨*dimension*⟩ qui est deux fois la taille naturelle de `\baselineskip`, sans étirement ni rétrécissement.

Exemple :

```
\count0 = 9\multiply \count0 by 8 \number\count0 ;
\divide \count0 by 12 \number\count0 \par
\skip0 = 20pt plus 2pt minus 3pt \multiply \skip0 by 3
Multiplied value of skip0 is \the\skip0.\par
\dimen0 = .5in \multiply\dimen0 by 6
\hbox to \dimen0{a\hfil b}
```

produit :

72; 6

Multiplied value of skip0 is 60.0pt plus 6.0pt minus 9.0pt.

a b

Terminer l'exécution

☞ `\bye`
Cette commande dit à TEX de remplir et produire la dernière page, d'imprimer tout insertions en suspens et de terminer l'exécution. C'est le moyen usuel de terminer votre fichier source.

`\end`
Cette commande dit à TEX de produire la dernière page et de terminer l'exécution. Elle ne remplit pas la page, néanmoins, dont il est normalement préférable d'utiliser `\bye` plutôt que `\end`.

Entrée et sortie

Opérations sur des fichiers d'entrée

☞ `\input` ⟨*nom de fichier*⟩
Cette commande demande à TEX de lire son entrée à partir du fichier ⟨*nom de fichier*⟩. Quand ce fichier est épuisé, TEX retourne lire à partir du source d'entrée précédent. Vous pouvez inclure des fichier d'entrée sur autant de niveau que vous voulez (dans les limites du raisonnable).

Quand vous saisissez un grand document, c'est souvent une bonne idée de structurer votre fichier principal comme une suite de commandes `\input` faisant référence aux parties subsidiaires du document. De cette façon, vous pouvez exécuter les parties individuelles facilement quand vous travaillez sur des épreuves. C'est aussi une bonne pratique de mettre toutes vos définitions de macro dans un fichier séparé et appeler ce fichier avec une commande `\input` comme première action de votre fichier principal.

TEX utilise des règles différentes pour balayer les noms de fichier que pour lire les tokens en général (voir p. 83). Si votre implémentation attend des nom de fichier avec extensions (habituellement précédées par un point), alors TEX procure l'extension par défaut `.tex`.

Exemple :
```
\input macros.tex
\input chap1 % equivalent to chap1.tex
```

`\endinput`
Cette commande dit à TEX d'arrêter de lire l'entrée du fichier courant quand il rencontrera la prochaine fin de ligne.

`\inputlineno`
Cette commande donne un nombre (pas une chaîne) donnant le numéro de ligne de la ligne courante, devant être le numéro qui apparaîtrait dans un message d'erreur si une erreur arrive à cette endroit.

`\openin` ⟨*nombre*⟩ `=` ⟨*nom de fichier*⟩
Cette commande demande à TEX d'ouvrir le fichier nommé ⟨*nom de fichier*⟩ et de le rendre lisible via le flot d'entrée désigné par ⟨*nombre*⟩. ⟨*nombre*⟩ doit être entre 0 et 15. Une fois que vous avez ouvert un fichier et l'avez connecté à un flot d'entrée, vous pouvez lire le fichier en utilisant la commande `\read` avec le numéro du flot d'entrée.

vous pouvez associer plus d'un flot d'entrée avec le même fichier. vous pouvez alors lire à plusieurs positions différentes dans le fichier, une pour chaque flot d'entrée.

Vous devez allouer les numéros de flots pour `\openin` en utilisant `\newread` (p. 252).

Exemple :
```
\newread\auxfile  \openin\auxfile = addenda.aux
% \auxfile now denotes the number of this opening
% of addenda.aux.
```

`\closein` ⟨*nombre*⟩
Cette commande demande à TEX de fermer le flot d'entrée numéroté ⟨*nombre*⟩, c'est-à-dire, clore l'association entre le flot d'entrée et son fichier. Le flot d'entrée avec ce numéro devient alors disponible pour être utiliser avec un fichier différent. Vous devez fermer un flot d'entrée une fois que vous avez fini d'utiliser son fichier.

Exemple :
```
\closein\auxfile
```

`\read` ⟨*nombre*⟩ `to` ⟨*séquence de contrôle*⟩
Cette commande demande à TEX de lire une ligne du fichier associé avec le flot d'entrée désigné par ⟨*nombre*⟩ et assigne les tokens de cette ligne à ⟨*séquence de contrôle*⟩. La séquence de contrôle devient alors une macro sans paramètre. Aucun développement de macro ne prend place durant l'opération de lecture. Si la ligne contient des accolades ouvrantes non appairées, TEX lira des lignes supplémentaires jusqu'a ce que les accolades soient toutes refermées. Si TEX atteint la fin du fichier sans fermer toutes les accolades, il se plaindra.

Si ⟨*nombre*⟩ est plus grand que 15 ou n'a pas été associé avec un fichier en utilisant `\openin`, TEX affichera l'incitation '⟨*séquence de contrôle*⟩ `=`' sur votre terminal et attendra que vous saisissiez une ligne d'entrée. Il assignera alors la ligne d'entrée à ⟨*séquence de contrôle*⟩. Si ⟨*nombre*⟩

est inférieur à zéro, il lira une ligne d'entrée de votre terminal mais omettra l'incitation.

Exemple :

```
\read\auxfile to \holder
% Expanding \holder will produce the line just read.
```

Opérations sur des fichiers de sortie

`\openout` ⟨*nombre*⟩ `=` ⟨*nom de fichier*⟩

Cette commande demande à TEX d'ouvrir le fichier nommé ⟨*nom de fichier*⟩ et le rend disponible pour l'écriture via le flux de sortie désigné par ⟨*nombre*⟩. ⟨*nombre*⟩ doit être entre 0 et 15. Une fois que vous avez ouvert un fichier et l'avez connecté à un flot de sortie, vous pouvez écrire dans le fichier en utilisant la commande `\write` avec le numéro du flot de sortie.

Un `\openout` génère un élément extraordinaire qui devient une partie d'une boîte. Le `\openout` ne prend pas effet avant que TEX n'envoi cette boîte dans le fichier `.dvi`, à moins que vous ayez fait précéder `\openout` de `\immediate`.

TEX ne se plaindra pas si vous associez plus d'un flux de sortie avec le même fichier, mais vous aurez un sacré bazar dans le fichier si vous essayez !

Vous devez allouer des numéro de flux pour `\openout` en utilisant `\newwrite` (p. 252).

Exemple :

```
\newwrite\auxfile  \openout\auxfile = addenda.aux
% \auxfile now denotes the number of this opening
% of addenda.aux.
```

`\closeout` ⟨*nombre*⟩

Cette commande demande à TEX de fermer le flot de sortie numéroté ⟨*nombre*⟩. c'est-à-dire, cesser l'association entre le flot de sortie et son fichier. Le flot de sortie avec ce numéro devient alors disponible pour être utilisé avec un fichier différent. Vous devez fermer un flot de sortie une fois que vous avez fini d'utiliser son fichier.

Un `\closeout` génère un élément extraordinaire qui devient une partie d'une boîte. Le `\closeout` ne prend pas effet avant que TEX n'envoie cette boîte dans le fichier `.dvi`, à moins que vous ayez fait précéder `\closeout` de `\immediate`.

Exemple :

```
\closeout\auxfile
```

`\write` ⟨*nombre*⟩ `{` ⟨*liste de token*⟩ `}`
Cette commande demande à TEX d'écrire ⟨*liste de token*⟩ dans le fichier associé avec le flot de sortie désigné par ⟨*nombre*⟩. Il génère un élément extraordinaire qui devient une partie d'une boîte. L'écriture actuelle ne prend pas de place tant que TEX n'envoie cette boîte dans le fichier `.dvi`, à moins que vous précédiez le `\write` par `\immediate`.

Pour un `\write` qui n'est pas immédiat, TEX ne développe pas de macros de ⟨*liste de token*⟩ tant que la liste de token n'est pas écrite dans le fichier. Les développements de macro suivent les mêmes règles que `\edef` (p. 238). En particulier, toute séquence de contrôle qui n'est pas le nom d'une macro est écrite comme un `\escapechar` suivi par le nom de la séquence de contrôle et un espace. tous les tokens '`#`' de ⟨*liste de token*⟩ sont doublés, c'est-à-dire, écrit comme '`##`'.

Si ⟨*nombre*⟩ n'est pas dans la fourchette de 0 à 15, TEX écrit ⟨*liste de token*⟩ dans le fichier log. Si ⟨*nombre*⟩ est supérieur à 15 ou n'est pas associé avec un flot de sortie, TEX écrit aussi ⟨*liste de token*⟩ sur le terminal.

Exemple :
```
\def\aa{a a}
\write\auxfile{\hbox{$x#y$} \aa}
% Writes the string ‘\hbox {$x##y$} a a’ to \auxfile.
```

`\immediate`
Cette commande doit précéder un `\openout`, un `\closeout` ou un `\write`. Elle demande à TEX d'exécuter l'opération de fichier spécifiée sans délai.

Exemple :
```
\immediate\write 16{I’m stuck!}
% has the same effect as \message
```

`\special` `{` ⟨*liste de token*⟩ `}`
Cette commande demande à TEX d'écrire ⟨*liste de token*⟩ directement dans le fichier `.dvi` quand il envera une page. Une utilisation typique de `\special` sera de dire au driver d'impression d'incorporer le contenu d'un fichier graphique nommé dans la page de sortie. La commande `\special` produit un élément extraordinaire qui associe ⟨*liste de token*⟩ avec une position particulière sur la page, plus précisément, la position qu'une boîte de taille nulle aurait eu si un telle boîte serait apparue à la place de la commande `\special`. Toute utilisation que vous devez faire de `\special` dépend strictement des drivers d'impression que vous avez à disposition.

Exemple :
```
\special{graphic expic}
% Display the graphics file ‘expic’ here.
```

`\newlinechar` [⟨*nombre*⟩ paramètre]
Ce paramètre contient un caractère qui indique une nouvelle ligne sur la sortie. Quand TEX rencontre ce caractère en lisant l'argument d'une commande `\write`, `\message` ou `\errmessage`, il débute une nouvelle ligne. Si `\newlinechar` n'est pas dans la fourchette 0–255, Il n'y a aucun caractère indiquant une nouvelle ligne en sortie. Plain TEX met `\newlinechar` à −1.

Exemple :
```
\newlinechar = `\^^J
\message{This message appears^^Jon two lines.}
```
produit dans la log :
```
This message appears
on two lines.
```

Voir aussi : `\newread`, `\newwrite` (p. 252).

Interpréter des caractères entrés

`\catcode` ⟨*charcode*⟩ [⟨*nombre*⟩ élément de table]
Cette entrée de table contient le code de catégorie du caractère dont le code ASCII est ⟨*charcode*⟩. Les codes de catégorie sont listé dans des page 56. En changeant le code de catégorie d'un caractère vous pouvez demande à TEX de traiter ce caractère différemment.

Exemple :
```
\catcode `\[ = 1 \catcode `\] = 2
% Make [ and ] act like left and right braces.
```

`\active`
Cette commande contient le code de catégorie pour un caractère actif, soit, le nombre 13.

Exemple :
```
\catcode `\@ = \active % Make @ an active character.
```

`\mathcode` ⟨*charcode*⟩ [⟨*nombre*⟩ élément de table]
Cette entrée de table contient le mathcode du caractère dont le code ASCII est ⟨*charcode*⟩ (voir "mathcode", p. 78). Le mathcode spécifie l'interprétation du caractère en mode mathématique.

Exemple :
```
\mathcode\> = "313E % as in plain TeX
% The > character has class 3 (relation), family 1 (math
% italic), and character code "3E
```

`\delcode` ⟨*charcode*⟩ [⟨*number*⟩ élément de table]
Cette entrée de table spécifie le code delimiter pour le caractère entrée dont le code ASCII est ⟨*charcode*⟩. Le code délimiteur dit à TEX comment trouver le meilleur caractère de sortie utiliser pour composer le caractère d'entrée indiqué comme délimiteur.

⟨*nombre*⟩ est normalement écrit en notation hexadécimale. Supposez que ⟨*nombre*⟩ soit le nombre hexadécimal $s_1s_2s_3\,l_1l_2l_3$. Alors, quand le caractère est utilisé comme un délimiteur, TEX prend le caractère ayant une petite variante $s_1s_2s_3$ et une grande variante $l_1l_2l_3$. Ici $s_1s_2s_3$ indique le caractère mathématique trouvé en position s_2s_3 de la famille s_1 et de même pour $l_1l_2l_3$. C'est la même convention que celle utilisée pour `\mathcode` (p. 259), sauf que `\mathcode` spécifie aussi une classe.

Exemple :
```
\delcode `( = "028300  % As in plain TeX.
```

`\endlinechar` [⟨*nombre*⟩ paramètre]
Ce paramètre contient le code de caractère pour le caractère que TEX attend à la fin de chaque ligne d'entrée. Une valeur qui ne serait pas dans le fourchette 0–255 indique qu'aucun caractère ne doit être attendu. Plain TEX laisse `\endlinechar` à `'\^^M` (le code ASCII pour ⟨return⟩).

`\ignorespaces`
Cette commande demande à TEX de lire et de développer des tokens jusqu'à ce qu'il en trouve un qui ne soit pas un token espace, ignorant tous les tokens espace qu'il trouve autrement. `\ignorespaces` est souvent pratique à la fin d'une macro comme un moyen de rendre la macro insensible à tout espace ou fin de ligne qui peut suivre son propre appel. (Une ligne vide après `\ignorespaces` continue à produire un token `\par`, de tout façon.)

Exemple :
```
\def\aa#1{yes #1\ignorespaces}
\aa{may}
be
```
produit :
yes maybe

Contrôler l'interaction avec TEX

`\errorstopmode`
Cette commande demande à TEX de stopper pour une interaction à chaque fois qu'il trouve une erreur. C'est le mode opératoire normal.

`\scrollmode`
Cette commande demande à TEX de ne pas stopper pour le plupart des erreurs, mais de continuer à afficher les messages d'erreur sur votre terminal. Saisir '`S`' ou '`s`' en réponse à un message d'erreur vous met en mode scroll.

`\nonstopmode`
Cette commande demande à TEX de ne pas stopper pour des erreurs, même celles réclamant des fichiers qu'il ne peut trouver, mais de continuer à afficher les messages d'erreur sur votre terminal. Saisir '`R`' ou '`r`' en réponse à un message d'erreur vous met en mode nonstop.

`\batchmode`
Cette commande demande à TEX de ne pas stopper pour des erreurs et de supprimer toute sortie sur votre terminal. Saisir '`Q`' ou '`q`' en réponse à un message d'erreur vous met en mode batch.

`\pausing` [⟨*nombre*⟩ paramètre]
Si ce paramètre est supérieur à zéro, TEX s'arrêtera à chaque ligne d'entrée pour vous donner une opportunité de la remplacer par une ligne différente. Si vous saisissez une modification, TEX utilisera cette ligne au lieu de l'originale ; si vous répondez par ⟨return⟩, TEX utilisera l'originale.

Mettre `\pausing` à 1 peut être pratique comme moyen de patcher un document quand TEX l'exécute. Par exemple, vous pouvez utiliser cette facilité pour insérer des commandes `\show` (voir plus loin).

Aide au diagnostique

Afficher des données internes

`\show` ⟨*token*⟩
`\showthe` ⟨*argument*⟩
`\showbox` ⟨*nombre*⟩
`\showlists`
Ces commandes enregistrent de l'information dans la log de votre exécution TEX :

- `\show` enregistre la signification de ⟨*token*⟩.
- `\showthe` enregistre quels tokens seront produit par `\the` ⟨*argument*⟩ (voir p. 242).
- `\showbox` enregistre le contenu du registre de la boîte numéroté ⟨*nombre*⟩. Le nombre de points affiché dans la log indique le nombre de niveaux d'inclusion de boîtes internes.

- `\showlists` enregistre le contenu de chaque liste que TEX est en train de construire. (les listes sont empilées les unes sur les autres.) Voir les pages 88–89 de *The TEXbook* et 99–99 de la traduction française pour plus d'information sur l'interprétation de la sortie de `\showlists`.

Pour `\show` et `\showthe`, TEX affiche aussi l'information sur votre terminal. Pour `\showbox` et `\showlists`, TEX n'affiche l'information sur votre terminal que si `\tracingonline` (p. 264) est supérieur à zéro ; si `\tracingonline` est à zéro ou moins (le cas par défaut), l'information n'est pas affichée.

à chaque fois que TEX rencontre une commande de type `\show` il stoppe pour un interaction. La requête d'interaction ne doit *pas* indiquer une erreur, mais doit vous donner l'opportunité de demander à TEX de vous montrer autre chose. Si vous ne voulez pas voir autre chose, pressez juste ⟨return⟩.

Vous pouvez contrôler la quantité de sortie produite par `\showbox` en mettant `\showboxbreadth` et `\showboxdepth` (p. 269). Ces paramètres ont des valeurs par défaut à 5 et 3 respectivement, C'est pourquoi juste cinq éléments apparaissent pour chaque boîtes décrites dans la sortie de log ci-dessous. (Le '`..etc.`' indique des éléments additionnels des boîtes qui ne sont pas affichés.)

Exemple :

```
\show a
\show \hbox
\show \medskip
\show &
```

produit dans la log :

```
> the letter a.
> \hbox=\hbox.
> \medskip=macro:
->\vskip \medskipamount .
> alignment tab character &.
```

Exemple :

```
\showthe\medskipamount
\toks27={\hbox{Joe's\quad\ Diner}}
\showthe\toks27
```

produit dans la log :

```
> 6.0pt plus 2.0pt minus 2.0pt.
> \hbox {Joe's\quad \ Diner}.
```

Exemple :

```
\setbox 3=\vbox{\hbox{A red dog.}\hrule A black cat.}
\showbox 3
```

produit dans la log :

```
> \box3=
\vbox(16.23332+0.0)x53.05565
.\hbox(6.94444+1.94444)x46.41675
..\tenrm A
..\glue 3.33333 plus 1.66498 minus 1.11221
..\tenrm r
..\tenrm e
..\tenrm d
..etc.
.\rule(0.4+0.0)x*
.\hbox(6.94444+0.0)x53.05565
..\tenrm A
..\glue 3.33333 plus 1.66498 minus 1.11221
..\tenrm b
..\tenrm l
..\tenrm a
..etc.
```

Exemple :

```
\vbox{A \hbox
   {formula
       $x \over y\showlists$}}
```

produit dans la log :

```
### math mode entered at line 3
\mathord
.\fam1 y
this will be denominator of:
\fraction, thickness = default
\\mathord
\.\fam1 x
### restricted horizontal mode entered at line 2
\tenrm f
\tenrm o
\tenrm r
\tenrm m
\kern-0.27779
\tenrm u
\tenrm l
\tenrm a
\glue 3.33333 plus 1.66666 minus 1.11111
spacefactor 1000
### horizontal mode entered at line 1
\hbox(0.0+0.0)x20.0
\tenrm A
\glue 3.33333 plus 1.66498 minus 1.11221
spacefactor 999
### internal vertical mode entered at line 1
prevdepth ignored
### vertical mode entered at line 0
prevdepth ignored
```

Voir aussi : `\showboxbreadth`, `\showboxdepth` (p. 269).

Spécifier ce qui est tracé

`\tracingonline` [⟨*nombre*⟩ paramètre]

Si ce paramètre est supérieur à zéro, TEX affichera les résultats de la trace (en incluant `\showbox` et `\showlists`) sur votre terminal en plus de les enregistrer dans le fichier log.

`\tracingcommands` [⟨*nombre*⟩ paramètre]

Si ce paramètre est égal à 1 ou plus, TEX enregistrera dans le fichier log plus de commandes qu'il exécute. Si `\tracingonline` est plus grand que

zéro, cette information apparaîtra aussi sur votre terminal. composer le premier caractère d'un mot compte comme une commande, mais (dans le but de la trace seulement) les actions de composition des caractères suivants et de toute ponctuation les suivant ne compte pas comme des commandes. Si `\tracingcommands` est à 2 ou plus, TEX enregistrera aussi des commandes qui sont développées plutôt qu'exécutées, par exemple, des tests conditionnels et leurs résultats.

Exemple :

```
\tracingcommands = 1 If $x+y>0$ we quit.\par
On the other hand, \tracingcommands = 0
```

produit dans la log :

```
{vertical mode: the letter I}
{horizontal mode: the letter I}
{blank space  }
{math shift character $}
{math mode: the letter x}
{the character +}
{the letter y}
{the character >}
{the character 0}
{math shift character $}
{horizontal mode: blank space  }
{the letter w}
{blank space  }
{the letter q}
{blank space  }
{\par}
{vertical mode: the letter O}
{horizontal mode: the letter O}
{blank space  }
{the letter t}
{blank space  }
{the letter o}
{blank space  }
{the letter h}
{blank space  }
{\tracingcommands}
```

`\tracinglostchars` [⟨*nombre*⟩ paramètre]

Si ce paramètre est plus grand que zéro, TEX enregistrera une indication dans le fichier log à chaque fois qu'il délaissera un caractère de sortie parce que ce caractère n'existe pas dans la police courante. Si `\tracingonline` est plus grand que zéro, cette information apparaîtra aussi sur votre terminal. Plain TEX le met par défaut à 1 (contrairement aux autres).

Exemple :

```
\tracinglostchars = 1
A {\nullfont few} characters.
```

produit dans la log :

```
Missing character: There is no f in font nullfont!
Missing character: There is no e in font nullfont!
Missing character: There is no w in font nullfont!
```

`\tracingmacros` [⟨*nombre*⟩ paramètre]

Si ce paramètre est à 1 ou plus, TEX enregistrera dans le fichier log le développement et les arguments de toutes les macros qu'il exécute. Si `\tracingmacros` est à 2 ou plus, TEX enregistrera, en plus, tout développement de listes de token telles que `\output` or `\everycr`. Si `\tracingonline` est plus grand que zéro, cette information apparaîtra aussi sur votre terminal.

Exemple :

```
\def\a{first \b, then \c}
\def\b{b} \def\c{c}
\tracingmacros = 2
Call \a once.
```

produit dans la log :

```
\a ->first \b , then \c

\b ->b

\c ->c
```

`\tracingoutput` [⟨*nombre*⟩ paramètre]

Si ce paramètre est supérieur à zéro, TEX enregistrera dans le fichier log le contenu de toutes les boîte qu'il envoie dans le fichier `.dvi`. Si `\tracingonline` est supérieur à zéro, cette information apparaîtra aussi sur votre terminal. Le nombre de points affiché dans la log indique le nombre de niveaux d'inclusion de boîtes sur cette ligne. Vous pouvez contrôler la quantité de trace en fixant `\showboxbreadth` et `\showboxdepth` (p. 269).

mettre `\tracingoutput` à 1 peut être très pratique quand vous essayez de déterminer pourquoi vous obtenez un espace supplémentaire sur une page.

Exemple :

```
% This is the entire file.
\tracingoutput = 1 \nopagenumbers
One-line page. \bye
```

produit dans la log :

```
Completed box being shipped out [1]
\vbox(667.20255+0.0)x469.75499
.\vbox(0.0+0.0)x469.75499, glue set 13.99998fil
..\glue -22.5
..\hbox(8.5+0.0)x469.75499, glue set 469.75499fil
...\vbox(8.5+0.0)x0.0
...\glue 0.0 plus 1.0fil
..\glue 0.0 plus 1.0fil minus 1.0fil
.\vbox(643.20255+0.0)x469.75499, glue set 631.2581fill
..\glue(\topskip) 3.05556
..\hbox(6.94444+1.94444)x469.75499, glue set 386.9771fil
...\hbox(0.0+0.0)x20.0
...\tenrm O
...\tenrm n
...\tenrm e
...\tenrm -
...etc.
..\glue 0.0 plus 1.0fil
..\glue 0.0 plus 1.0fill
.\glue(\baselineskip) 24.0
.\hbox(0.0+0.0)x469.75499, glue set 469.75499fil
..\glue 0.0 plus 1.0fil
```

`\tracingpages` [⟨*nombre*⟩ paramètre]

Si ce paramètre est supérieur à zéro, TEX enregistrera dans le fichier log ses calculs sur le coût des différentes coupures de page qu'il essaie. Si `\tracingonline` est supérieur à zéro, cette information apparaîtra aussi sur votre terminal. TEX produit une ligne de cette sortie à chaque fois qu'il commence à placer une boîte ou une insertion sur la liste de page courante, et aussi à chaque fois qu'il exécute une point de coupure potentiel pour la page. Examiner cette sortie peut être utile quand vous essayez de déterminer la cause d'une mauvaise coupure de page. Voir les pages 112–114 de *The TEXbook* et 999–999 de la traduction française pour une illustration et une explication de cette sortie.

Certaines distributions de TEX ignore la valeur de `\tracingpages` ainsi elles peuvent s'exécuter plus rapidement. Si vous devez utiliser ce paramètre, soyez certain d'utiliser une distribution qui lui répond.

`\tracingparagraphs` [⟨*nombre*⟩ paramètre]

Si ce paramètre est supérieur à zéro, TEX enregistrera dans le fichier log ses calculs sur le coût des différentes coupures de ligne qu'il essaie. Si `\tracingonline` est supérieur à zéro, cette information apparaîtra aussi sur votre terminal. TEX produit cette sortie quand il atteint la fin de chaque paragraphe. Voir les pages 98–99 de *The TEXbook* et 99–

99 de la traduction française pour une illustration et une explication de cette sortie.

Certaines distributions de TEX ignore la valeur de `\tracingparagraphs` ainsi elles peuvent s'exécuter plus rapidement. Si vous devez utiliser ce paramètre, soyez certain d'utiliser une distribution qui lui répond.

`\tracingrestores` [⟨*nombre*⟩ paramètre]

Si ce paramètre est supérieur à zéro, TEX enregistrera dans le fichier log les valeurs qu'il restitue quand il rencontre la fin d'un groupe. Si `\tracingonline` est supérieur à zéro, cette information apparaîtra aussi sur votre terminal.

Certaines distributions de TEX ignore la valeur de `\tracingrestores` ainsi elles peuvent s'exécuter plus rapidement. Si vous devez utiliser ce paramètre, soyez certain d'utiliser une distribution qui lui répond.

`\tracingstats` [⟨*nombre*⟩ paramètre]

Si ce paramètre est à 1 ou plus, TEX incluera un rapport sur les ressources qu'il utilise pour exécuter votre travail (voir la page 300 de *The TEXbook* et 999 de la traduction française pour une liste et l'explication de ces ressources). Du reste, si `\tracingstats` est à 2 ou plus, TEX fera un rapport sur ses utilisation de mémoire à chaque fois qu'il fera un `\shipout` (p. 154) pour une page. Le rapport apparaît à la fin du fichier log. Si `\tracingonline` est supérieur à zéro, cette information apparaîtra aussi sur votre terminal. Si vous avez des problèmes avec TEX dépassant une de ses capacités, l'information procurée par `\tracingstats` peut vous aider à mettre le doigt sur la cause de vos difficulté.

Certaines distributions de TEX ignore la valeur de `\tracingstats` ainsi elles peuvent s'exécuter plus rapidement. Si vous devez utiliser ce paramètre, soyez certain d'utiliser une distribution qui lui répond.

L'exemple suivant montre un extrait de la sortie de trace que vous obtenez sur une distribution de TEX. Elle peut être différente sur d'autres distributions.

Exemple :

```
\tracingstats=1
```

produit dans la log :

```
Here is how much of TeX's memory you used:
 4 strings out of 5540
 60 string characters out of 72328
 5956 words of memory out of 262141
 921 multiletter control sequences out of 9500
 14794 words of font info for 50 fonts, out of 72000 for 255
 14 hyphenation exceptions out of 607
 7i,4n,1p,68b,22s stack positions out of 300i,40n,60p,3000b,4000s
```

`\tracingall`
Cette commande demande à TEX de brancher toutes les formes de trace disponibles. Elle met aussi `\tracingonline` à 1 pour que la sortie de trace apparaisse sur votre terminal.

`\showboxbreadth` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie le nombre maximum d'éléments de liste que TEX affiche pour un niveau d'une boîte quand il produit la sortie pour `\showbox` ou `\tracingoutput`. Plain TEX met `\showboxbreadth` à 5.

`\showboxdepth` [⟨*nombre*⟩ paramètre]
Ce paramètre spécifie le niveau de la plus profonde liste que TEX affiche quand il produit la sortie pour `\showbox` ou `\showlists`. Plain TEX met `\showboxdepth` à 3.

Envoyer des messages

`\message {` ⟨*liste de token*⟩ `}`
`\errmessage {` ⟨*liste de token*⟩ `}`
Cette commandes affiche le message donné par ⟨*liste de token*⟩ sur votre terminal et l'entre aussi dans la log. Toutes les macros du message sont développées, mais aucune commandes n'est exécutées. C'est la même règle que TEX utilise pour `\edef` (p. 238).

Pour `\errmessage`, TEX fait une pause de la même façon qu'il le ferait pour un de ses propres messages d'erreur et affiche les tokens `\errhelp` si vous demandez de l'aide.

Vous pouvez générer des messages multi-ligne en utilisant le caractère `\newlinechar` (p. 259).

Exemple :
```
\message{Starting a new section.}
```

`\wlog {` ⟨*liste de token*⟩ `}`
Cette commande écrit ⟨*liste de token*⟩ dans le fichier log. TEX développe ⟨*liste de token*⟩ en accord avec les mêmes règles qu'il utilise pour `\edef` (p. 238).

Exemple :
```
\wlog{Take two aspirins and call me in the morning.}
```
produit dans la log :
```
Take two aspirins and call me in the morning.
```

`\errhelp` [⟨*liste de token*⟩ paramètre]
Ce paramètre contient la liste de token que TEX affiche quand vous demandez de l'aide en réponse à une commande `\errmessage`. Nous

vous recommandons quand vous générez un message d'erreur avec `\errmessage`, de mettre `\errhelp` à une chaîne qui décrit la nature de l'erreur et utiliser `\newhelp` pour produire cette chaîne. Vous pouvez utiliser le caractère `\newlinechar` pour produire des messages multi-ligne.

`\newhelp` ⟨*séquence de contrôle*⟩ `{` ⟨*texte d'aide*⟩ `}`
Cette commande assigne le message d'aide donné par ⟨*texte d'aide*⟩ à ⟨*séquence de contrôle*⟩. Il procure un moyen efficace de définir le texte d'aide qui complète un message d'erreur. Avant d'émettre le message d'erreur avec la commande `\errmessage`, vous devez assigner ⟨*séquence de contrôle*⟩ à `\errhelp`. Le texte d'aide apparaîtra alors si l'utilisateur saisi '`H`' ou '`h`' en réponse au message d'erreur.

Exemple :

```
\newhelp\pain{Your input includes a token that I find^^J
   to be offensive. Don't bother me again with this^^J
   document until you've removed it.}
\errhelp = \pain \newlinechar = `\^^J
% ^^J will start a new line
\errmessage{I do not appreciate receiving this token}
```

produit dans la log :

```
! I do not appreciate receiving this token.
l.8 ...t appreciate receiving this token.}

? H
\Your input includes a token that I find
 to be offensive. Don't bother me again with this
 document until you've removed it.
```

`\errorcontextlines` [⟨*nombre*⟩ paramètre]
Ce paramètre détermine le nombre de paires de ligne de contexte, sans compter la première et la dernière, que TEX imprime quand il rencontre une erreur. En le mettant à 0 vous pouvez vous débarrasser de long messages d'erreur. Vous pouvez forcer le contexte entier en saisissant quelque chose comme :

```
I\errorcontextlines=100\oops
```

en réponse à une erreur, puisque la séquence de contrôle non définie `\oops` causera une autre erreur. Plain TEX met `\errorcontextlines` à 5.

Voir aussi : `\write` (p. 258), `\escapechar` (p. 234).

Initialiser TEX

`\dump`
Cette commande, qui ne doit pas apparaître à l'intérieur d'un groupe, met en réserve le contenu de la mémoire de TEX dans un fichier format (p. 69). En utilisant `virtex`, une forme spéciale "vierge" de TEX, vous pouvez alors recharger le fichier format à grande vitesse et continuer dans le même état que TEX était au moment du dump. `\dump` termine aussi une exécution. Puisque `\dump` ne peut être utilisé qu'avec `initex`, pas en format de production de TEX, elle n'est utile qu'aux gens qui installent TEX.

`\everyjob` [⟨*liste de token*⟩ paramètre]
Ce paramètre contient une liste de token que TEX développe au début de chaque exécution. Parce qu'un assignement à `\everyjob` ne peut affecter l'exécution courante (au moment où vous avez fait l'assignement, c'est déjà trop tard), elle n'est utile qu'ux gens qui préparent des fichiers de format.

10 Trucs et astuces

TEX est une programme complexe qui travaille parfois selon de ses propres envies mystérieuses. Dans cette section nous vous offrons quelques trucs pour résoudre les problèmes que vous pourrez rencontrer et expliquer quelques techniques pratiques.

Corriger de mauvaises coupures de page

Parfois TEX coupe une page juste au milieu du matériel que vous voulez garder ensemble.—par exemple, une entête de section et le texte qui la suit ou une courte suite d'items liés. Il y a deux moyens de corriger la situation :

- Vous pouvez forcer le matériel à rester ensemble.
- Vous pouvez forcer une coupure de page à un endroit différent.

La plus simple façon de forcer TEX a garder le matériel ensemble sur une page est de l'englober dans un vbox en utilisant la commande `\vbox` (p. 167). Une vbox est normalement meilleure qu'une hbox pour faire cela parce que le plus souvent, le matériel à garder ensemble, par exemple, une suite de paragraphe, sera du matériel en mode vertical. Vous pouvez précéder et suivre le vbox par une commande de paragraphe implicite ou explicite (soit une ligne blanche, soit une `\par`) ; autrement TEX pourrait essayer de faire la partie vbox sur un paragraphe adjacent. La méthode vbox a une limitation importante : vous ne pouvez pas l'appliquer à une portion de texte inférieure a un paragraphe.

Vous pouvez parfois garder les lignes d'un paragraphe ensemble en l'englobant dans un groupe et en donnant à `\interlinepenalty` (p. 144) la valeur 10000 au début du groupe (ou n'importe où avant la fin du paragraphe). Cette méthode demandera à TEX de considérer les

coupures de page dans ce paragraphe comme étant infiniment indésirable. Néanmoins ; si toutes les coupures de page que TEX peut trouver sont infiniment indésirables, il coupera la page dans le paragraphe quand même.

Une commande `\nobreak` (p. 142) après la fin d'un paragraphe empêche TEX de couper la page sur l'item suivant (a moins que cet item provoque un penalty inférieur à 10000). Ces aussi la meilleure façon d'empêcher une coupure de page après une entête, puisque une entête réagit normalement comme un paragraphe. Le `\nobreak` doit suivre la ligne blanche ou le `\par` qui termine la paragraphe pour que TEX ne traite pas le `\nobreak` comme faisant partie du paragraphe. Pour que le `\nobreak` soit effectif, il doit aussi être placé avant tout point de coupure légal à l fin du paragraphe. Le ressort que TEX insère avant le paragraphe suivant est comme un point de coupure, et ainsi pour tout ressort vertical que vous insérez explicitement après un paragraphe. Ainsi, le `\nobreak` doit normalement être la toute première chose après la fin du paragraphe ou de l'entête.

Vous pouvez utiliser la commande `\eject` (p. 143) pour forcer TEX à couper une page à un endroit particulier. Dans un paragraphe, vous pouvez utiliser la combinaison '`\vadjust{\vfill\eject}`' (p. 126) pour forcer une coupure après la prochaine ligne de sortie complète. La raison de faire précéder `\eject` par `\vfill` (p. 163) est d'obliger TEX à remplir la page avec un espace blanc. De toute façon, utiliser `\eject` pour résoudre des problèmes de coupure de page a un désavantage majeur : si les frontières de page de votre document changent, les coupures de page que vous avez insérées peuvent ne plus être là où vous les voulez.

Si vous ne procurez pas à TEX une commande `\vfill` pour remplir la page après un `\eject`, TEX redistribue l'espace blanc excédentaire le mieux qu'il peut et alors se plain généralement d'un "underfull `\vbox` (badness 10000) has occurred while `\output` is active." Vous pouvez rencontrer un problème similaire avec toutes les méthodes mentionnées ci-dessus pour englober du matériel que vous voulez garder lié.

La commande `\filbreak` (p. 143) procure un moyen de garder les ligne d'un ou plusieurs paragraphes (ou autre matériel de mode vertical) lié sur une page. Si vous englobez un paragraphe entre `\filbreak`, TEX ignorera effectivement les `\filbreak` si le paragraphe débute sur la page courante et coupe la page avant le premier `\filbreak` si le paragraphe ne débute pas. Si vous mettez des `\filbreak` autour de chaque paragraphe

dans une séquence de paragraphes, comme ceci :

```
\filbreak
⟨paragraph⟩
\filbreak
⟨paragraph⟩
\filbreak
  ⋮
⟨paragraph⟩
\filbreak
```

TEX gardera les lignes de chaque paragraphe lié sur une page. Si TEX coupe une page sur un `\filbreak`, i remplira le bas de la page avec un espace blanc.

Parfois vous pourrez demander à TEX de modifier la longueur d'une page en changeant le paramètre `\looseness` (p. 130) pour un ou plusieurs paragraphes. Rendre un `\looseness` négatif dans un paragraphe fait que TEX essaye de séparer le paragraphe en moins de lignes ; le rendre positif fait que TEX essaye d'étendre le paragraphe en plus de lignes. Le désavantage de changer `\looseness` est que l'espacement inter-mot de la région affecté ne sera pas optimal. Vous pouvez obtenir plus d'information sur le façon dont TEX appréhende les coupures de ligne en mettant `\tracingpages` (p. 267) à 1.

Préserver la fin d'une page

Parfois vous avez besoin de modifier quelque chose sur une seule page et vous voulez éviter de réimprimer tout le document. Si vos modifications ne changent pas trop la pas, il y a de l'espoir. Vous devez modifier la fin de la page pour qu'elle tienne au même endroit ; les méthodes sont similaire à celles pour modifier une mauvaise coupure de page.

Si la fin de la page originale se trouve entre des paragraphes, vous pouvez forcer une coupure de page au même endroit en utilisant toutes les méthodes que nous avons décrit plus haut. Autrement, vous devez forcer une coupure de ligne *et* une coupure de page à un endroit particulier. Si la nouvelle page est plus courte que l'ancienne, la séquence :

```
\vadjust{\vfill\eject}\break
```

devrait faire l'affaire. Mais si la nouvelle page est plus longue, le problème est plus difficile parce que TEX a probablement déjà compressé la page aussi fermement qu'il le peut. Vos seuls espoirs dans ce cas sont de mettre `\looseness` (p. 130) à une valeur négative, pour raccourcir certain sauts verticaux sur la page, pour ajouter un peu de rétrécissement `\parskip`

(p. 147) s'il était différent de zéro, ou, en dernier recours, diminuer `\baselineskip` (p. 139) encore plus fermement.

Garder de l'espace en haut d'une page

Vous pouvez utiliser habituellement la commande `\vskip` (p. 161) pour garder de l'espace vertical sur une page. Cela ne marche pas en haut d'une page, toutefois, car TEX abandonne le ressort, les crénages et les pénalités qui arrivent juste après une coupure de page. Utilisez la commande `\topglue` (p. 162) à la place ; il produit un ressort qui ne disparaît jamais.

Corriger de mauvaise coupure de ligne

Si TEX coupe une ligne au milieu du matériel que vous voulez garder sur une seule ligne, il y a plusieurs moyens de corriger la situation :

- Vous pouvez forcer une coupure à un endroit proche avec la commande `\break` (p. 126).
- Vous pouvez insérer un tilde (`~`) entre deux mots (see p. 111) pour empêcher une coupure entre eux
- Vous pouvez signaler à TEX des césures qu'il ne considèrerait pas autrement en insérant uns ou plusieurs césures optionnelles dans plusieurs mots (voir `\-`, p. 132).
- Vous pouvez englober plusieurs mots dans une hbox en utilisant la commande `\hbox` (p. 166).

Le désavantage de toutes ces méthodes, à par l'insertion de césures optionnelles, est qu'elles rendent impossible pour TEX de trouver un jeu de coupures de ligne satisfaisant. Si cela arrive néanmoins, TEX trouvera une ou plusieurs boîtes trop ou pas assez pleines et s'en plaindra. La méthode hbox a un désavantage de plus : parce que TEX voit une hbox comme une entité simple sans considérer son contexte, l'espace entre les mots dans la hbox peut ne pas être cohérent avec celui du reste de la ligne.

Corriger des boîtes trop ou pas assez pleines

Si TEX se plaint d'une boîte trop pleine, cela signifie que vous avez mis plus de matériel dans une boîte qu'elle ne peut en contenir. Similairement, si TEX se plaint d'une boîte pas assez pleine, cela signifie que vous n'avez

pas mis assez de matériel dans la boîte. Vous pouvez rencontrer ces plaintes dans de nombreuses circonstances différentes, donc, regardons les plus commune :

- Une boîte trop pleine d'une ligne d'un paragraphe indique que la ligne était trop longue et que TEX ne peut pas réarranger le paragraphe pour rendre la ligne plus courte. Si vous mettez `\emergencystretch` (p. 129) à une valeur différente de zéro, cela peut résoudre le problème en autorisant TEX à mettre plus d'espace entre les mots. Une autre solution est de mettre `\tolerance` (p. 128) à 10000, mais c'est un peu comme émettre des lignes avec beaucoup trop d'espace. Sinon, une autre solution est d'insérer une césure optionnelle dans un mot critique Que TEX ne sait pas couper. Si tout cela ne suffit pas, vous pourriez essayer de réécrire le paragraphe. Une solution qui est rarement satisfaisante est d'augmenter `\hfuzz` (p. 176), autorisant ainsi TEX de construire des lignes qui dépasse dans la marge droite.
- Une boîte pas assez pleine d'une ligne d'un paragraphe indique que la ligne était trop courte et que TEX ne peut pas réarranger le paragraphe pour rendre la ligne plus longue. TEX formera de telles ligne en étirant ses espaces inter-mots au delà de leurs limites normales. Deux des solutions pour des lignes trop pleines mentionnées ci-dessus s'appliquent aussi aux lignes pas assez pleines : insérer des césures optionnelles et réécrire le paragraphe. Des lignes pas assez pleines ne vous gênerons pas si vous utilisez un format justifié à gauche, que vous pourrez obtenir avec la commande `\raggedright` (p. 122).
- La plainte :

  ```
  Underfull \vbox (badness 10000) has occurred
     while \output is active
  ```

  indique que TEX n'a pas assez de matériel pour remplir une page. La cause probable est que vous avez utiliser des vbox pour garder du matériel ensemble et TEX a rencontrer une vbox près du bas d'une page qui ne peut pas rentrer sur cette page. Il a mis la vbox sur la page suivante, mais en ce faisant a laissé trop d'espace vide dans la page courante. Dans ce cas vous pouvez soit insérer plus d'espace quelque part sur la page courante, soit couper la vbox en plus petites parties.

  Une autre cause possible de cette plainte est d'avoir un long paragraphe qui occupe une page entière sans coupure. Puisque TEX ne fait pas varier l'espacement entre les lignes, il peut être incapable de remplir un trou au bas de la page d'un montant d'une fraction de l'espacement de la ligne. Cela peut arriver si `\vsize` (p. 146), la longueur de la page, n'est pas un multiple de `\baselineskip` (p. 139), l'espace entre des lignes de bases consécutive.

Sinon, une autre cause de cette plainte, similaire à la précédente, est d'avoir mis `\parskip` (p. 147), le ressort inter-paragraphes à une valeur qui n'a pas assez d'étirement ou de rétrécissement. Vous pouvez résoudre ces deux derniers problèmes en augmentant `\vfuzz` (p. 176).

- La plainte

```
Overfull \vbox (296.30745pt too high) has occurred
   while \output is active
```

indique que vous avez construit une vbox qui est plus longue que la page. vous devriez juste la faire plus courte.
- Les seule solutions pour une hbox ou une vbox trop pleine que vous avez construit avec les commandes `\hbox` or `\vbox` (pp. 166, 167) sont de sortir quelque chose de la boîte, insérer un ressort négatif avec `\hss` ou `\vss` (p. 164) ou d'augmenter la taille de la boîte.
- Si vous rencontrez une hbox ou une vbox pas assez pleine que vous avez construit avec `\hbox` ou `\vbox`, vous feriez mieux normalement de remplir la boîte avec `\hfil` ou `\vfil` (p. 163).

Retrouver des espaces entre-mots perdus

Si vous trouvez que TEX a trop rapproché deux mots, la cause courante est une séquence de contrôle qui a absorbé l'espace qui la suit. Mettre un espace contrôlé (\␣) après la séquence de contrôle.

éviter des espaces entre-mots non désirés

Si vous obtenez un espace dans votre document où vous ne vouliez et ne pensiez pas en avoir, la cause la plus courante, d'après notre expérience, est une fin de ligne ou un espace après une accolade. (Si vous faites de jolies choses avec les codes de catégories, vous avez introduit beaucoup d'autres causes tout aussi sympathiques.) Normalement, TEX traduit une fin de ligne pas un espace, et considère un espace après une accolade ouvrante ou fermante comme significatif.

Si l'espace non désiré est causé par un espace après une accolade dans une ligne d'entrée, retirez-le. Si l'espace non désiré est causé par une accolade à la fin d'une ligne d'entrée, mettez une '`%`' immédiatement après l'accolade. Le '`%`' débute un commentaire, mais ce commentaire ne nécessite aucun texte.

Une définition de macro peut aussi introduire des espaces non désirés si vous ne les avez pas écrites soigneusement. Si vous obtenez des espaces non désirés quand vous appelez une macro, vérifiez sa définition pour être

sûr que vous n'avez pas un espace non désiré après une accolade ou que vous n'avez pas terminé une ligne de la définition immédiatement après une accolade. On termine souvent une ligne de définition de macro après une accolade pour rendre la définition plus lisible. Par sécurité, mettez un '`%`' après toute accolade qui termine une ligne de définition de macro. Il n'est peut-être pas nécessaire, mais il ne fera pas de mal.[1]

Si vous ave du mal à localiser la source d'un espace non désiré, essayez de mettre `\tracingcommands` (p. 264) à 2. Vous obtiendrez une commande `{blank space}` dans le fichier log pour chaque espace que voit TEX.

Cela aide de connaître les règles d'espace de TEX :

1) Des espaces sont ignorés au début des lignes d'entrée.
2) Des espaces en fin de lignes d'entrée sont ignoré en *toute* circonstance, bien que la fin de ligne elle-même soit traitée comme un espace. (une ligne complètement blanche, néanmoins, génère un token `\par`.)
3) De multiple espaces sont traités comme un espace simple, mais seulement si elles apparaissent ensemble dans votre entrée. Ainsi un espace suivant les arguments d'un appel de macro n'est pas combiné avec l'espace final produit par l'appel de macro. à la place, vous obtenez deux espaces.
4) Des espaces sont ignorés après des mots de contrôle.
5) Des espaces sont de fait ignorés après des nombres, des dimensions et le '`plus`' et le '`minus`' des spécifications de ressort.[2]

Si vous avez changé le code de catégorie de l'espace ou du caractère fin de ligne, oubliez tout cela.

éviter un excès d'espace autour d'un affichage

Si vous obtenez trop d'espace au dessus d'un affichage mathématique, c'est peut être parce que vous avez laissé une ligne blanche dans votre entrée avant l'affichage. La ligne blanche débute un nouveau paragraphe et mets TEX en mode vertical. Quand TEX voit un '`$`' en mode vertical, il rebascule vers le mode horizontal et insère le ressort inter-paragraphes (`\parskip`) suivie par le ressort inter-lignes (`\baselineskip`). Ensuite, quand il débute l'affichage lui-même, il insère *plus* de ressort (soit `\abovedisplayskip`, soit `\abovedisplayshortskip`, selon la longueur de la ligne précédente). Ce dernier ressort est le seul que vous désirez. Pour éviter d'obtenir un ressort inter-paragraphe ainsi, ne laissez pas

[1] Il faut reconnaître qu'il n'y a que de rares cas où vous avez réellement d'une fin de ligne après une accolade.

[2] En réalité, TEX n'ignore qu'un seul espace à cet endroit. Puisque des espaces multiples sont habituellement réduit à un seul espace, pourtant, l'effet est d'ignorer tout nombre d'espaces.

de ligne blanche au dessus des affichages mathématiques ou autrement terminez un paragraphe (avec `\par`, disons) juste avant un affichage mathématique.

Similairement, si vous obtenez trop d'espace après un affichage mathématique, c'est peut-être parce que vous avez laissé une ligne blanche dans votre entrée après l'affichage. retirez le simplement.

éviter un excès d'espace après un paragraphe

Si vous obtenez trop d'espace vertical après un paragraphe qui a été produit par une macro, vous devez avoir eu un ressort inter-paragraphe produite par la macro, un paragraphe vide, et ensuite encore un ressort inter-paragraphe. Vous pouvez débarrasser du saut du second paragraphe en insérant :

```
\vskip -\parskip
\vskip -\baselineskip
```

juste après l'appel de la macro. Si vous avez toujours ce problème avec une certaine macro, vous pouvez mettre ces lignes à la fin de la définition de macro à la place. Vous pourriez aussi résoudre le problème en ne laissant jamais de ligne blanche après l'appel de macro—si vous voulez une ligne blanche simplement pour rendre votre saisie plus lisible, commencez la par un '`%`'.

Changer la forme du paragraphe

Plusieurs paramètres de TEX—`\hangindent`, `\leftskip`, etc.—affectent la façon dont TEX forme des paragraphes et les coupe en lignes. Ces paramètres sont utilisés indirectement dans des commandes plain TEX telles que `\narrower` et `\hang` ; vous pouvez aussi les assigner directement. Si vous avez utilisé une de ces commandes (ou changé un de ces paramètres), mais que le changement de commande ou de paramètre ne semble pas avoir le moindre effet sur un paragraphe, le problème peut être que vous avez terminé un groupe avant d'avoir terminé le paragraphe. Par exemple :

```
{\narrower She very soon came to an open field, with
a wood on the other side of it: it looked much darker
than the last wood, and Alice felt a little timid
about going into it.}
```

Ce paragraphe ne sera par composé en resserré parce que l'accolade fermante termine le groupe `\narrower` avant que TEX ait eu une chance

de couper le paragraphe en lignes. à la place, mettez un `\par` avant l'accolade fermante ; ensuite vous obtiendrez l'effet que vous voulez.

Mettre des paragraphes dans une boîte

Supposez que vous ayez quelques paragraphes de texte que vous voulez mettre à un endroit particulier sur la page. La manière évidente de faire cela est d'englober les paragraphes dans une hbox d'une taille appropriée et ensuite de placer la hbox où vous voulez qu'elle soit. Hélas, la manière évidente ne marche pas parce que TEX ne fait pas de coupure de ligne dans le mode horizontal restreint. Si vous l'essayez, vous obtiendrez un message d'erreur trompeur vous suggérant que vous avez oublié la fin d'un groupe. Le moyen de contourner cette restriction est d'écrire :

`\vbox{\hsize =` ⟨*dimen*⟩ ... ⟨*paragraphs*⟩ ...`}`

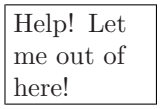
où ⟨*dimen*⟩ est la longueur de ligne que vous voulez pour les paragraphes. C'est ce dont vous avez besoin, en particulier, quand vous voulez englober des paragraphes dans une boîte (une boîte entourée de traits droits, pas une boîte TEX).

dessiner des lignes

Vous pouvez utiliser les commandes `\hrule` et `\vrule` (p. 178) pour tracer des lignes, c'est-à-dire, des filets. Vous devez savoir (a) où vous pouvez utiliser chaque commandes et (b) comment TEX détermine les longueurs des filets quand vous n'avez pas donné les longueurs explicitement.

- Vous pouvez utiliser `\hrule` quand TEX est dans un mode vertical et `\vrule` quand TEX est dans un mode horizontal. Ces règles signifies que vous ne pouvez mettre un filet horizontal dans une hbox ou un filet vertical dans une vbox. Vous pouvez, néanmoins, construire un filet horizontal qui semble vertical en spécifiant les trois dimensions et en le rendant grand et fin. De même, vous pouvez construire un filet vertical qui semble horizontal en le rendant court, plat et allongé.
- Un filet horizontale à l'intérieure d'une vbox a la même largeur que la vbox si vous n'avez pas donné la largeur du filet explicitement. Des filets verticaux à l'intérieur de hbox réagissent de manière analogue. Si vos filets ressortent trop long ou trop court, réglez les dimensions de la boîte englobante.

Comme exemple, supposez que vous vouliez produire :



L'entrée suivante le fera :

```
\hbox{\vrule
   \vbox{\hrule \vskip 3pt
      \hbox{\hskip 3pt
          \vbox{\hsize = .7in \raggedright
             \noindent Help! Let me out of here!}%
      \hskip 3pt}%
   \vskip 3pt \hrule}%
\vrule}
```

Nous devons mettre le texte dans un vbox de façon à ce que TEX l'exécute comme un paragraphe. Les quatre niveaux d'emboîtement sont réellement nécessaire—si vous en doutez, essayez d'exécuter cet exemple avec moins de niveaux.

Créer des entêtes ou des pieds de page multi-lignes

Vous pouvezz utiliser les commandes `\headline` et `\footline` (p. 149) pour produire des entêtes et des pieds de page, mais elles ne marchent pas proprement pour des entêtes et des pieds de page de plus d'une ligne. Néanmoins, vous pouvez obtenir des entêtes et des pieds de page multi-lignes en redéfinissant certaine des macros subsidiaire dans la routine de sortie de TEX.

Pour un entête multi-lignes, vous devez faire trois choses :

1) Redéfinir la macro `\makeheadline` qui est appelée de la routine de sortie de TEX.
2) Augmenter `\voffset` du montant d'espace vertical consommé par les lignes en plus.
3) Diminuer `\vsize` du même montant.

L'exemple suivant montre comment vous pourriez faire cela :

```
\advance\voffset by 2\baselineskip
\advance\vsize by -2\baselineskip
\def\makeheadline{\vbox to 0pt{\vss\noindent
   Header line 1\hfil Page \folio\break
   Header line 2\hfil\break
   Header line 3\hfil}%
   \vskip\baselineskip}
```

Vous pouvez normalement suivre le modèle de cette définition, substituez simplement vos propres lignes d'entête et choisissez un multiple

de `\baselineskip` approprié (un de moins que le nombre de lignes dans l'entête).

Pour un pied de page multi-lignes, la méthode est similaire :

1) Redéfinissez la macro `\makefootline` qui est appelée de la routine de sortie de TEX.
2) Diminuez `\vsize` du montant d'espace vertical consommé par les lignes en plus.

L'exemple suivant montre comment vous pouvez faire cela :

```
\advance\vsize by -2\baselineskip
\def\makefootline{%
   \lineskip = 24pt
   \vbox{\raggedright\noindent
   Footer line 1\hfil\break
   Footer line 2\hfil\break
   Footer line 3\hfil}}
```

Encore une fois, vous pouvez normalement suivre le modèle de cette définition. La valeur de `\lineskip` détermine le montant d'espace entre la ligne de base de la dernière ligne du texte principal sur la page et la ligne de base de la première ligne du pied de page.

Trouver des accolades orphelines

La plupart de temps, quand votre entrée TEX souffre d'accolades non concordantes, vous obtenez un diagnostic de TEX assez proche de l'endroit où vous avez réellement fait l'erreur. Mais une des erreurs les plus frustrantes que vous puissiez subir d'une exécution de TEX, juste avant que TEX s'arrête, est la suivante :

```
(\end occurred inside a group at level 1)
```

Cela indique qu'il y a une accolade ouvrante en trop ou une accolade fermante manquante quelque part dans votre document, mais il ne vous donne aucune indication du tout sur l'endroit où le problème arrive. Donc comment pouvez vous trouver ?

Un truc de débuggage que nous trouvons pratique est d'insérer la ligne suivante ou son équivalente à cinq ou six endroit également espacés dans le document (et pas dans un groupe connu) :

```
}% une fausse fermante
```

supposons que le problème soit une accolade ouvrante en trop. Si elle est, disons entre la troisième et la quatrième fausse fermante, vous obtiendrez des messages d'erreur des trois premières fausses fermantes mais pas de la quatrième. La raison est que TEX ignorera les trois première fausses fermantes après s'en être plaint, mais la quatrième fausse fermante appariera l'accolade ouvrante en trop. Ainsi vous savez que l'accolade

ouvrante en trop est quelque part entre la troisième et la quatrième fausse fermante. Si la région de l'erreur est encore trop large pour que vous la trouviez, enlevez simplement le jeu original de fausse fermantes et répétez le processus dans cette région. Si le problème est un accolade fermante manquante plutôt qu'une ouvrante en trop, vous serez capable de la découvrir quand vous aurez trouvé sa compagne.

Cette méthode ne marche pas dans toutes les circonstances. En particulier, elle ne marche pas si votre document consiste en plusieurs groupes réellement grand. Mais souvent vous pourrez trouver des variations de cette méthode qui vous mènerons vers cette accolade insaisissable.

Si tout le reste échoue, essayez de raccourcir votre entrée en retirant la dernière moitié du fichier (après sauvegarde de la version originale d'abord !) ou en insérant une commande `\bye` au milieu. Si l'erreur persiste, vous saurez qu'elle est dans la première moitié ; Si ça tourne, vous saurez qu'elle est dans la seconde. En répétant ce processus vous trouverez éventuellement l'erreur.

Fixer des dimensions

La manière la plus simple de définir une dimension est de la spécifier directement, c'est-à-dire :

```
\hsize = 6in
```

Vous pouvez aussi spécifier une dimension en terme d'autres dimensions ou comme un mélange de différentes unités, mais c'est du boulot. Il y a deux façons de construire une dimension comme une combinaison :

1) Vous pouvez ajouter une dimension à un paramètre de dimension ou à un registre de dimension. Par exemple :

   ```
   \hsize = 6in \advance\hsize by 3pc % 6in + 3pc
   ```

2) Vous pouvez indiquer une dimension comme un multiple d'un paramètre ou un registre de dimension ou de ressort. Dans ce cas, TEX convertit le ressort en une dimension en sacrifiant l'étirement et le rétrécissement. Par exemple :

   ```
   \parindent = .15\hsize
   \advance\vsize by -2\parskip
   ```

Créer des polices composites

Il est parfois utile de creer une "police composite", nommée par une séquence de contrôle $\mathcal{F}$, dans laquelle tous les caractères sont pris d'une

police f_1 sauf pour certain qui sont emprunté d'une autre police f_2. Vous pourrez alors mettre du texte dans la police composite en utilisant $\mathcal{F}$ juste comme vous utilisez toute autre identifiant de police.

Vous pouvez créer une telle police composite en définissant $\mathcal{F}$ comme une macro. Dans la définition de $\mathcal{F}$, vous sélectionnez d'abord la police f_1 et ensuite définissez des séquences de contrôle qui produisent les caractères empruntés, mis dans f_2. Par exemple, supposons que vous voulez créer une police composite `\britrm` qui a tous les caractères de `cmr10` sauf pour le signe dollar, pour lequel vous voulez emprunter le symbole de la livre sterling de la police `cmti10`. Le symbole de la livre sterling de `cmti10` apparait à la même position de police que le signe dollar dans `cmr10`. Voici comment faire :

```
\def\britrm{%
   \tenrm % \tenrm names the cmr10 font
   \def\${{\tenit\char `\$}}% \tenit names the cmti10 font.
}
```

Maintenant à chaque fois que vous lancerez la police appelée `\britrm`, `\$` produira un symbole de livre sterling.

Vous pouvez aussi avoir le même effet en changeant les codes de catégorie des caractères en question pour rendre ces caractères actifs et ensuite procurer une définition pour le caractère. Par exemple :

```
\catcode `* = \active
\def*{{\tentt \char `\*}}
```

Dans ce cas l'astérisque sera pris de la police `\tentt`. Si cous saisissez alors la ligne d'entrée :

```
Debbie was the * of the show.
```

elle sera composée ainsi :

Debbie was the * of the show.

Reproduire du texte verbatim

Du texte verbatim est du texte qui est reproduit dans un document composé exactement comme il apparaît dans son entrée. L'usage le plus commun du texte verbatim est dans la composition d'entrée informatique, incluant les programmes pour ordinateurs et les entrées de TEX lui-même. l'entrée informatique n'est pas simple à reproduire pour deux raisons :

1) Certain caractères (symboles de contrôle, caractères d'échappement, accolades, etc.) ont une signification spéciales en TEX.
2) Les fins de ligne et les espaces multiples sont traduit en espaces simples.

Dans le but de produire du texte verbatim, vous devez effacer les significations spéciales et débrancher la traduction. C'est beaucoup mieux fait par des macros.

Pour effacer les signification spéciales, vous devez changer les codes de catégorie des caractères qui ont des significations spéciales. La macro suivante illustre comment vous devez le faire :

```
\chardef \other = 12
\def\deactivate{%
   \catcode`\\ = \other   \catcode`\{ = \other
   \catcode`\} = \other   \catcode`\$ = \other
   \catcode`\& = \other   \catcode`\# = \other
   \catcode`\% = \other   \catcode`\~ = \other
   \catcode`\^ = \other   \catcode`\_ = \other
}
```

Mais attention ! Une fois que vous avez changé les codes de catégorie de ce façon, vous perdez la faculté d'utiliser des séquences de contrôle puisqu'il n'y a plus de caractère d'échappement. Vous avez besoin d'un moyen de revenir dans le mode normal d'opération. Nous expliquerons comment faire cela dans un moment, après avoir considéré l'autre problème : débrancher la traduction des espaces et des fins de ligne.

Plain TEX a deux commandes qui ensemble résolve en partie le problème : `\obeyspaces` (p. 113) et `\obeylines` (p. 128). Les deux choses qu'elles ne font pas sont de préserver les espaces au début d'une ligne et de préserver les lignes blanches. Pour cela vous avez besoin de mesures plus fortes—qui sont apportées par la macro `\obeywhitespace` que nous sommes sur le point de définir.

TEX normalement insiste pour collecter des lignes par paragraphes. Un moyen de le convaincre de prendre littéralement les bords des lignes est de transformer les lignes individuelles en paragraphes.[3] Vous pouvez faire cela en redéfinissant le caractère de fin de ligne pour produire la séquence de contrôle `\par`. Les trois définitions de macro suivantes montrent comment :

```
\def\makeactive#1{\catcode`#1 = \active \ignorespaces}
{% The group delimits the text over which ^^M is active.
   \makeactive\^^M %
   \gdef\obeywhitespace{%
   % Use \gdef so the definition survives the group.
      \makeactive\^^M %
      \let^^M = \newline %
      \aftergroup\removebox % Kill extra paragraph at end.
      \obeyspaces %
   }%
}
```

[3] Un autre moyen est de transformer le caractère de fin de ligne en une commande `\break` et d'ajouter un ressort infini à la fin de chaque ligne.

```
\def\newline{\par\indent}
\def\removebox{\setbox0=\lastbox}
```

Un point subtil de la définition de `\obeywhitespace` est que `^^M` doit être rendu actif à la fois quand `\obeywhitespace` est *définie* et quand elle est *utilisée*.

Pour être capable de revenir à des opérations normales après du texte verbatim, vous devez choisir un caractère qui apparaît rarement voire pas du tout dans le texte verbatim. Ce caractère sert comme caractère d'échappement temporaire. La barre verticale (`|`) est parfois un bon choix. Avec ce choix, les macros :

```
\def\verbatim{\par\begingroup\deactivate\obeywhitespace
   \catcode `\| = 0 % Make | the new escape character.
}

\def\endverbatim{\endgroup\endpar}

\def\|{|}
```

fera l'affaire. Dans le texte verbatim, vous pouvez utiliser une double barre verticale (`||`) pour en faire une, et vous terminez le texte verbatim avec `|endverbatim`.

Il y a plusieurs variations de cette technique :

- Si un langage informatique a des mots-clés, vous pouvez convertir chaque mot-clé en une commande qui compose ce mot-clé en caractère gras. Chaque mot-clé de l'entrée doit être alors précédé par le caractère d'échappement.
- Si vous avez un caractère (encore, supposons qu'il s'agisse de la barre verticale) qui n'apparaît *jamais* dans le texte verbatim, vous pouvez le rendre actif et lui faire finir le texte verbatim. Les définitions de macro ressemblent alors à cela :

```
{\catcode `\| = \active
\gdef\verbatim{%
   \par\begingroup\deactivate\obeywhitespace
   \catcode `| = \active
   \def |{\endgroup\par}%
}}
```

Les idées présentées ici ne procurent qu'une simple approche de la composition de programmes informatiques. Le reproduction verbatim n'est souvent pas aussi révélatrice ou lisible qu'une version qui utilise des conventions typographiques qui reflètent la syntaxe et même les sémantiques du programme. Si vous désirez poursuivre ce sujet plus loin, nous vous recommandons le livre suivant :

Baecker, Ronald M., and Marcus, Aaron, *Human Factors and Typography for More Readable Programs*. Reading, Mass.: Addison-Wesley, 1990.

Utiliser des macros externes

Si TEX se plaint d'une "séquence de contrôle interdite [forbidden control sequence]", vous avez probablement utilisé une macro externe dans un contexte non-externe (voir "outer", p. 84). Une macro externe est une macro dont la définition est précédée par `\outer`. Une macro externe ne peut pas être utilisée dans un argument de macro, dans une définition de macro, dans le préambule d'un alignement ou dans un texte conditionnel, c'est-à-dire, un texte qui ne sera développé que quand un test conditionnel aura un résultat particulier. Certaines macros ont été définie comme externe parce qu'elles n'ont pas de raison d'être utilisées dans ces contextes et qu'un tel usage est probablement une erreur. Le seul moyen de contourner ce problème est de redéfinir la macro ou de déplacer son utilisation dans un contexte acceptable.

Utiliser une macro externe dans un contexte impropre peut aussi provoquer que TEX se plaigne d'un "runaway situation" ou d'un "incomplete conditional". Le problème peut être difficile à diagnostiquer parce que le message d'erreur ne donne aucun indice ni ce qui se passe. Si vous obtenez un tel message d'erreur, recherchez autour un appel à une macro externe. Vous ne pouvez pas toujours savoir si une macro particulière est externe, mais la commande '`\show\a`' (p. 261) vous montrera la définition de `\a` et vous dira aussi si `\a` est externe.

Changer des codes de catégorie

Il est parfois pratique de faire des changements locaux au code de catégorie d'un caractère à certaine partie de votre document. Par exemple, vous devez composer une programme informatique ou quelque chose d'autre qui utilise normalement des caractères actifs pour un usage spécial. Vous voudrez alors désactiver ces caractères pour que TEX les traite comme étant comme tout autre caractère.

Si vous faites un tel changement local au code de catégorie d'un caractère, vous serez parfois consterné de trouver que TEX semble ne prêter aucune attention quelconque a votre changement. Deux aspects du comportement de TEX sont vraisemblablement en cause :

1) TEX détermine le code de catégorie d'un caractère entrée et l'attache au caractère quand il le lit. Supposons que vous lisiez un tilde (`~`) et qu'ensuite vous changiez le code de catégorie des tildes, mais fassiez

le changement avant que l'estomac de TEX ait réellement exécuté de tilde *particulier* (voir "Anatomie de TEX", p. 48). TEX répondra encore que ce tilde utilise le code de catégorie comme il était avant le changement. Cette difficulté survient typiquement quand le tilde fait partie d'un argument d'une macro et que la macro elle-même change le code de catégorie du tilde.

2) Quand TEX assortie un appel d'une macro à la définition de cette macro, il n'assortie pas seulement les caractères du modèle de paramètre mais aussi leurs codes de catégorie. Si le code de catégorie d'un modèle caractère n'est pas égal au code de catégorie du même caractère dans l'appel, TEX ne considèrera pas les caractères comme concordant. Cet effet peut produire de mystérieux résultats parce que il *fait* comme si le modèle concorde. Par exemple, si vous avez défini une macro :

```
\def\eurodate#1/#2/#3{#2.#1.#3}
```

alors le caractère slash devra avoir le même code de catégorie quand vous appelez `\eurodate` que quand vous avez défini `\eurodate`.

Si le problème surgit parce que le caractère gênant est un argument d'une macro, alors le remède usuel est de redéfinir la macro comme une paire de macros `\mstart` et `\mfinish`, où `\mstart` doit être appelée avant le texte d'argument et `\mfinish` doit être appelé après. `\mstart` alors initialise les codes de catégorie et `\mfinish` défait le changement, peut-être simplement en finissant un groupe.

Faire des fichiers de macro plus lisibles

Vous pouvez faire un fichier de macros plus lisible en mettant les code de catégories de l'espace à 9 (caractère ignoré) et `\endlinechar` (p. 260) à −1 au début du fichier. Vous pourrez alors utiliser des espaces et des fins de lignes librement dans les définitions de macro sans obtenir d'espaces indésirables quand vous appelez les macros. Les caractères ignorés ne génèreront pas d'espaces, mais agiront encore comme terminateurs pour des séquences de contrôle. Si vous voulez réellement un espace, vous pouvez encore l'obtenir avec la commande `\space` (p. 111).

Bien sûr vous devrez restaurez les codes de catégories de l'espace et de la fin de ligne à leurs valeurs normales (10 et 5, respectivement) à la fin du fichier. Vous pouvez faire cela soit en englobant tout le fichier dans un groupe soit en restaurant les valeurs explicitement. Si vous choisissez d'englober le fichier dans un groupe, vous devrez alors aussi mettre `\globaldefs` à 1 pour que toutes les définitions de macro soient globales et donc visible en dehors du groupe.

Un exemple miniature d'un fichier de macro de cette forme est :

```
\catcode `\  = 9  \endlinechar = -1
```

```
\def \makeblankbox #1 #2 {
   \hbox{\lower \dp0 \vbox{\hidehrule {#1} {#2}
   \kern -#1 % overlap rules
   \hbox to \wd0{\hidevrule {#1} {#2}%
      \raise \ht0  \vbox to #1{} % vrule height
      \lower \dp0 \vtop to #1{}  % vrule depth
      \hfil \hidevrule {#2} {#1} }
   \kern -#1 \hidehrule {#2} {#1} } }

\def\hidehrule #1 #2 {
   \kern -#1 \hrule height#1 depth#2 \kern -#2 }
\def\hidevrule #1 #2 {
   \kern -#1 {\dimen0 = #1 \advance \dimen0 by #2
   \vrule width \dimen0 } \kern -#2 }

\catcode `\  = 10  \endlinechar = `\^^M
```

Sans les changement de code de catégorie, ces macros aurait été écrite de manière beaucoup plus compacte, utilisant moins d'espaces et plus de '%' à la fin des lignes.

11 Comprendre les messages d'erreur

L'interprétation des messages d'erreur des TEX peut parfois être comparée à aller chez votre médecin avec la sensation d'être fatigué et obtenir, en réponse, une anomalie de votre chimie sanguine. C'est probablement l'explication de votre détresse, mais il n'est pas facile de la décrire. Quelques règles simples vous feront aller plus loin en vous aidant à comprendre les messages d'erreur de TEX et à en obtenir d'avantage.

Votre premier objectif sera de comprendre ce que vous avez fait pour que TEX se plaigne. Le second (si vous travaillez interactivement) sera de déceler le plus d'erreurs que vous pouvez en une seule fois.

Regardons un exemple. Supposez que votre source contienne la ligne :

```
We skip \quid a little bit.
```

Vous pensiez saisir '`\quad`', mais vous avez tapé '`\quid`' à la place. Voici la réponse que vous obtiendrez de TEX:

```
 !
Undefined control sequence.
l.291 We skip \quid
                  a little bit.
?
```

Ce message apparaîtra sur votre terminal et dans votre fichier `.log`. La première ligne, qui commence toujours par un point d'exclamation (`!`), vous indique ce qu'est le problème. Les deux dernières lignes avant le prompt '`?`'(qui sont, dans ce cas-ci, également, les deux lignes suivante) vous indiquent où était TEX quand il a trouvé l'erreur. Il a trouvé l'erreur sur la ligne 291 du fichier source courant, et la coupure entre les deux lignes de message indique la position précise de TEX dans la ligne 291, à savoir, juste après `\quid`. Le fichier source courant est celui juste après la parenthèse gauche la plus récemment ouverte dans le résultat de votre exécution sur votre terminal (voir les p. 9).

Cette erreur particulière, un "undefined control sequence", est l'une de la plus commune que vous puissiez obtenir. Si vous répondez au prompt par un autre '`? `', TEX affichera le message suivant :

```
Type <return> to proceed, S to scroll future error messages,
R to run without stopping, Q to run quietly,
I to insert something, E to edit your file,
1 or ... or 9 to ignore the next 1 to 9 tokens of input,
H for help, X to quit.
```

Voici ce que signifient ces propositions :

- Si vous saisissez ⟨return⟩, TEX continuera à traiter votre document. Dans ce cas, il ignorera juste le `\quid`.
- Si vous saisissez '`S`' (ou '`s`'—majuscule et minuscule sont équivalentes ici), TEX traitera votre document sans s'arrêter, *sauf* s'il rencontre un fichier absent. Les messages d'erreur continuerons à apparaître sur votre terminal et dans le fichier `.log`.
- Si vous saisissez '`R`' ou '`r`', vous obtiendrez le même effet que pour '`S`', sauf que TEX ne s'arrêtera pas pour les fichiers manquants.
- Si vous saisissez '`Q`' ou '`q`', TEX continuera à traiter votre document mais ne s'arrêtera pas pour des erreurs ni ne les montrera sur votre terminal. Les erreurs apparaîtront toujours dans le fichier `.log`.
- Si vous saisissez '`X`' ou '`x`', TEX nettoiera ce qu'il peut au mieux, jettera la page sur laquelle il travaillait et stoppera. Vous pourrez imprimer ou regarder les pages que TEX a déjà traitées.
- Si vous saisissez '`E`' ou '`e`', TEX nettoiera et se terminera comme il le fait pour '`X`' ou '`x`' et entrera alors dans votre éditeur de texte, vous plaçant sur la ligne incorrecte. (tous les systèmes ne supportent pas cette option.)
- Si vous saisissez '`H`' ou '`h`', vous obtiendrez une autre explication de l'erreur montrée sur votre terminal et probablement du conseil sur quoi faire à son sujet. Cette explication apparaîtra également dans votre fichier `.log`. Pour le "undefined control sequence" ci-dessus, vous obtiendrez :

  ```
  The control sequence at the end of the top line
  of your error message was never \def'ed. If you have
  misspelled it (e.g., `\hobx'), type `I' and the correct
  spelling (e.g., `I\hbox'). Otherwise just continue,
  and I'll forget about whatever was undefined.
  ```

- Si vous saisissez '`?`', vous recevrez encore ce même message.

Les deux autres solutions de rechange, saisir '`I`' ou un petit nombre entier, fournissent des moyen de demander à TEXde revenir en arrière

pour que votre erreur ne cause pas d'autres erreurs plus tard dans votre document :

- Si vous saisissez '`I`' ou '`i`' suivi de texte, alors TEX insérera ce texte comme s'il s'était produit juste après le point d'erreur, au niveau les plus secrets où le TEX travaille. Dans le cas de l'exemple ci-dessus, cela signifie à la position de TEX dans votre source original, c'est-à-dire, juste après '`\quid`'. Plus tard vous verrez un exemple qui montre la différence entre insérer quelque chose au niveau les plus secrets et l'insérer dans votre source original. Dans l'exemple ci-dessus d'"undefined control sequence", si vous saisissez :

  ```
  I\quad
  ```

  TEX effectuera la commande `\quad` et produira un espace cadratin là où vous avez eu l'intention d'avoir un.
- Si vous saisissez un nombre entier positif inférieur à 100 (pas inférieur à 10 comme le message le suggère maladroitement), TEX supprimera ce nombre de tokens du niveau les plus secrets où il travaille. (si vous saisissez un nombre entier supérieur ou égal à 100, TEX supprimera 10 tokens !)

Voici un autre exemple d'erreur commune :

```
Skip across \hskip 3cn by 3 centimeters.
```

Le message d'erreur pour ceci est :

```
! Illegal unit of measure (pt inserted).
<to be read again>
                   c
<to be read again>
                   n
l.340 Skip across \hskip 3cn
                              by 3 centimeters.
```

Dans ce cas, TEX a vu que '`3`' est suivi par quelque chose qui n'est pas une unité de mesure reconnue, et ainsi suppose que l'unité de mesure est le point. TEX relira le token '`cn`' et l'insérera dans votre saisie, ce qui n'est pas ce que vous voulez. Dans ce cas, vous pouvez obtenir un meilleur résultat en saisissant d'abord '`2`' pour reculer avant '`cn`'. Vous recevrez le message :

```
<recently read> n

l.340 Skip across \hskip 3cn
                            by 3 centimeters.
```

Maintenant vous pouvez saisir '`I\hskip 3cm`' pour obtenir le saut que vous vouliez (en plus du saut de `3pt` que vous avez déjà obtenu).[1]

[1] en saisissant '`I\unskip\hskip 3cm`' vous pouvez vous débarrasser du saut de `3pt`.

Si vous saisissez quelque chose qui n'est valide qu'en mode mathématique, TEX s'orientera vers le mode mathématique pour vous que ce soit ce que vous avez vraiment voulu ou non. Par exemple :

```
So \spadesuit s are trumps.
```

Voici le message d'erreur de TEX :

```
! Missing $ inserted.
<inserted text>
                $
<to be read again>
                   \spadesuit
l.330 So \spadesuit
                  s are trumps.
```

Puisque le symbole `\spadesuit` n'est permis qu'en mode mathématique, TEX a inséré un '`$`' devant lui. Après que TEX ait inséré un token, il se place *devant* ce token, dans ce cas, le '`$`', prêt à le lire. saisir '`2`' fera que TEX sautera le '`$`' et le token '`\spadesuit`', le laissant prêt à traiter le '`s`' de '`s are trumps`'. (si vous laissez juste TEX continuer, il composera '`s are trumps`' en mode mathématique.)

Voici un exemple où le diagnostic d'erreur de TEX est complètement erroné :

```
\hbox{One \vskip 1in two.}
```

Le message d'erreur est :

```
! Missing } inserted.
<inserted text>
                }
<to be read again>
                   \vskip
l.29 \hbox{One \vskip
                     1in two.}
```

Le problème est que vous ne pouvez pas utiliser `\vskip` quand TEX est en mode horizontal restreint, c'est-à-dire, en construisant un hbox. Mais au lieu de rejeter le `\vskip`, TEX a inséré une accolade droite devant lui afin d'essayer de fermer le hbox. Si vous acceptez la correction de TEX, TEX se plaindra encore quand il arrivera plus tard à l'accolade droite correcte. Il se plaindra également à propos de tout ce qui précède cette accolade droite et qui n'est pas permis en mode vertical. Ces plaintes répétitives seront particulièrement embrouillantes parce que les erreurs qu'elles indiquent sont fausses, elles sont le résultat des effets propagés de l'insertion inadéquate de l'accolade droite. Votre meilleur pari est de saisir '`5`', et de sauter après tous les tokens de '`}\vskip 1in`'.

Voici un exemple similaire dans lequel le message d'erreur est plus long que ce que nous avons vus jusqu'ici :

```
\leftline{Skip \smallskip a little further.} But no more.
```

L'erreur ici est que `\smallskip` ne fonctionne qu'en mode vertical. Le message d'erreur est quelque chose comme :

```
! Missing } inserted.
<inserted text>
                }
<to be read again>
                   \vskip
\smallskip ->\vskip
                    \smallskipamount
<argument> Skip \smallskip
                              a little further.
\leftline #1->\line {#1
                       \hss }
l.93 ...Skip \smallskip a little further.}
                                                But no more.
```

Les messages d'erreur ici vous donnent un tour d'horion des macros qui sont utilisées dans l'implémentation de `\leftline` dans plain TEX—des macros auxquelles vous ne vous inquiétez probablement pas. La première ligne vous indique que TEX prévoit de traiter le problème en insérant une accolade droite. TEX n'a pas encore réellement lu l'accolade droite, ainsi vous pouvez la supprimer si vous le choisissez. Chaque composant du message après la première ligne (celle avec le '!') occupe une paire de lignes. Voici ce que signifient les paires de lignes successives :

1) La première paire indique que TEX a inséré, mais pas encore lu, une accolade droite :
2) La paire suivante indique qu'après avoir lu l'accolade droite, TEX relira un '`\vskip`' (obtenue de la macro définition du `\smallskip`).
3) La troisième paire indique que TEX développait la macro `\smallskip` quand il a trouvé l'erreur. ces deux lignes montrent également la définition de `\smallskip` et indique où TEX est parvenu en développant et en exécutant cette définition. Spécifiquement, il a juste essayée sans succès d'exécuter la commande `\vskip`. En général, une ligne diagnostique qui commence par une commande suivi de '`->`' indique que TEX a développé et exécuté une macro de ce nom.
4) La quatrième paire indique que TEX traitait un argument de macro quand il a trouvé le `\smallskip` et indique également la position de TEX du fait de l'argument, c'est-à-dire, qu'il a juste traité le `\smallskip` (sans succès). En regardant vers la prochaine paire de lignes nous pouvons voir que l'argument a été passé à `\leftline`.
5) La cinquième paire indique que TEX développait la macro `\leftline` quand il a trouvé l'erreur. (dans cet exemple l'erreur s'est produite pendant que TEX était en train d'interpréter plusieurs définitions de macro à différents degrés de développement.) Sa position après `#1` indique que la dernière chose qu'il a vu était le premier (et dans ce cas-ci le seul) argument de `\leftline`.

6) La dernière paire indique où TEX est placé dans votre fichier source. Notez que cette position est bien au delà de la position où il a insèré l'accolade droite et relu le '`\vskip`'. C'est parce que TEX a déjà lu tout l'argument du `\leftline` de votre fichier source, même s'il n'a traité qu'une partie de cet argument. Les points au début de la paire indiquent qu'une partie précédente de la ligne source n'est pas montrée. Cette partie précédente, en fait, inclut la commande `\leftline` qui a rendu le `\vskip` illégal.

Dans un long message comme ceci, vous ne trouverez généralement utile que la première ligne et la dernière paire de lignes ; mais cela aide parfois de savoir ce que sont les autres lignes. N'importe quel texte que vous insérerez ou effacerez sera inséré ou supprimé au niveau les plus secrets. Dans cet exemple l'insertion ou la suppression se produira juste avant l'accolade droite inséré. Notez en particulier que dans ce cas-ci, TEX met tout texte que vous pourriez insérer, *non* dans votre texte source mais dans une définition de macro plusieurs niveaux plus bas. (naturellement, la définition de macro originale n'est pas modifiée.)

Vous pouvez employer la commande `\errorcontextlines` (p. 270) pour limiter le nombre de paires de lignes de contexte d'erreur que TEX produit. Si vous n'êtes pas intéressé par toute l'information que TEX vous fournit, vous pouvez placer `\error contextlines` à 0. Cela vous ne donnera que les premières et dernières paires de lignes.

En conclusion, nous mentionnerons deux autres indicateurs qui peuvent apparaître au début d'une paire de lignes de message d'erreur :

- `<output>` indique que TEX était au milieu de sa routine de sortie quand cette erreur s'est produite.
- `<write>` indique que TEX était en train d'exécuter une commande `\write` quand cette erreur s'est produite. TEX détectera une telle erreur quand il fera réellement le `\write` (pendant un `\shipout`), et non quand il rencontre le `\write`.

12 Un abrégé de macros utiles

Cette section décrit `eplain.tex`, une collection de macros et autres définitions qui prolongent le plain TEX. Les noms des divers macros expliquent leurs buts, mais habituellement n'expliquent pas comment elles fonctionnent, ou ne fournissent pas de détails explicites sur la façon dont les employer. cette information est contenue dans les fichiers source de `eplain.tex` et dans la documentation qui l'accompagne. Voir "Ressources" (p. 18) pour la façon d'obtenir `eplain.tex`.

Préliminaires

Nous commençons par quelques macros pour des codes de catégorie qui changent et des définitions pratiques pour deux macros utilisées fréquemment.

```
\def\makeactive#1{\catcode`#1 = \active \ignorespaces}%
\chardef\letter = 11 \chardef\other = 12
\def\uncatcodespecials{%
   \def\do##1{\catcode`##1 = \other}%
   \dospecials}% Defined in plain.
```

Pour définir '`^^M`' comme caractère actif, vous devez englober la définition dans un groupe et appeler quelques moyens supplémentaires. La macro `\letreturn` vous permet de définir '`^^M`' sans ce code supplémentaire (que vous le pouvez voir dans la définition ci- dessous).

```
{\makeactive\^^M \long\gdef\letreturn#1{\let^^M = #1}}%
```

Ces macros consomment un, deux, ou trois arguments.

```
\def\gobble#1{}\def\gobbletwo#1#2{}%
\def\gobblethree#1#2#3{}%
```

Maintenant nous établissons quelques conventions pour lire le reste du fichier. Dans le fichier nous autorisons des séquences de contrôle "privés" qui contiennent '@' dans leurs noms. Ces séquences de contrôle ne sont pas accessible de l'extérieur de ce fichier (à moins que vous rechangez le code de catégorie du '@').

```
\catcode`@ = \letter    % Allow control sequences with @.
\let\@plainwlog = \wlog % Don't log register allocations.
\let\wlog = \gobble
\newlinechar = `^^J
```

Les deux macros suivantes fournissent un formulation pratiques de résultat de diagnostique. `\loggingall` déclenche tout le traçage, mais ne provoque le résultat de la trace que dans le fichier `.log`et pas sur votre terminal. `\tracingboxes` montre complètement les boîtes quand elles sont tracées. (TEX ne montre normalement que trois niveaux de boîte et cinq items dans chaque boîte.)

```
\def\loggingall{\tracingcommands\tw@\tracingstats\tw@
   \tracingpages\@ne\tracingoutput\@ne
   \tracinglostchars\@ne\tracingmacros\tw@
   \tracingparagraphs\@ne\tracingrestores\@ne
   \showboxbreadth\maxdimen\showboxdepth\maxdimen}%
\def\tracingboxes{\showboxbreadth = \maxdimen
   \showboxdepth = \maxdimen}%
```

L'épaisseur par défaut des traits est de 0.4 pt. Vous pouvez produire des traits de n'importe quelle épaisseur par défaut que vous voulez en la redéfinissant `\vruledefaultwidth`, `\hruledefaultheight` et `\hruledefaultdepth` et en employant `\ehrule` et `\evrule` au lieu de `\hrule` et `\vrule`. (le 'e' représente "eplain".) Si vous donnez une dimension explicite (par exemple, `\ehrule height 16pt`), TEX l'utilisera.

```
\newdimen\hruledefaultheight  \hruledefaultheight = 0.4pt
\newdimen\hruledefaultdepth   \hruledefaultdepth = 0.0pt
\newdimen\vruledefaultwidth   \vruledefaultwidth = 0.4pt
\def\ehrule{\hrule height\hruledefaultheight
   depth\hruledefaultdepth}%
\def\evrule{\vrule width\vruledefaultwidth}%
```

La convention `\%` pour écrire un caractère '%' ne marche pas quand vous voulez inclure ce caractère dans la token liste de `\write`. Vous pouvez utiliser `\percentchar` pour réaliser cela. Nous redéfinissons également `^^L` comme étant sans effet pour que vous puissiez l'utiliser dans une définition de macro ou un argument.

```
{\catcode`\% = \other \gdef\percentchar{%}}%
 \def^^L{\par
}%
```

`\tokstostring` convertit son argument en liste de tokens de caractère. Il n'emploie que des développements qui sont manipulées dans l'œsophage

de TEX. Cette propriété est lui nécessaire pour travailler avec `\edef`. Elle est utilisée par les macros de références croisées (p. 312).

Pour Fractionner l'argument sur l'espace, nous devons utiliser deux macros subsidiaires. `\@ttsA` trouve les espaces et `\@ttsB` manipule une séquence de token sans aucun espace. Chaque espace est remplacé par le développement de `\spacesub`.

```
\def\tokstostring#1{\@ttsA#1 \ttsmarkA}%
\def\@ttsA#1 #2\ttsmarkA{\ifempty{#1}\else
      \@ttsB #1\@ttsmarkB
      \ifempty{#2}\else
         \spacesub\@ttsA#2\ttsmarkA\fi\fi}%
\def\@ttsB#1{\ifx #1\@ttsmarkB\else
      \string #1%
      \expandafter\@ttsB\fi}%
\def\@ttsmarkB{\@ttsmarkB}% should never be expanded
\def\spacesub{+}%
```

`\ifempty` teste si son argument est vide.

```
\def\ifempty#1{\@ifempty #1\@emptymarkA\@emptymarkB}%
\def\@ifempty#1#2\@emptymarkB{\ifx #1\@emptymarkA}%
\def\@emptymarkA{\@emptymarkA}%
```

La macro `\for` implémente une version TEX du "for loop" des langages de programmation traditionnel. ces macros viennent directement de LATEX.

```
\def\for#1:=#2\do#3{\edef\@fortmp{#2}%
   \ifx\@fortmp\empty \else
      \expandafter\@forloop#2,\@nil,\@nil\@@#1{#3}\fi}%
\def\@nnil{\@nil}%
\def\@fornoop#1\@@#2#3{}%
\def\@forloop#1,#2,#3\@@#4#5{\def#4{#1}\ifx #4\@nnil
   \else #5\def#4{#2} ifx #4\@nnil \else
      #5\@iforloop #3\@@#4{#5}\fi\fi}%
\def\@iforloop#1,#2\@@#3#4{\def#3{#1}\ifx #3\@nnil
   \let\@nextwhile=\@fornoop \else #4\relax
      \let\@nextwhile=\@iforloop\fi
         \@nextwhile#2\@@#3{#4}}%
```

`\obeywhitespace` est utile pour reproduire des coupures de ligne, des interlignes et des espaces dans votre source. Il combine les effets de `\obeylines` et `\obeyspaces`, et met également des espaces au début d'une ligne devant être imprimée. Les caractères "tabulation" ne sont pas affectés ; ils produisent toujours un ressort normal.

```
\def\alwaysspace{\hglue\fontdimen2\the\font \relax}%
{\makeactive\^^M \makeactive\ %
\gdef\obeywhitespace{%
\makeactive\^^M\def^^M{\par\indent}%
```

```
\aftergroup\@removebox% Kill extra paragraph at end.
\makeactive\ \let =\alwaysspace}}%
\def\@removebox{\setbox0=\lastbox}
```

`\frac` est une bonne façon d'afficher des fractions dans du texte quand vous ne voulez pas utiliser `\over` et que mettre simplement "1/2" ne semble pas beau. Cette macro est la réponse à Exercise 11.6 de *The TEXbook* et de la traduction française.

```
\def\frac#1/#2{\leavevmode
   \kern.1em \raise .5ex \hbox{\the\scriptfont0 #1}%
   \kern-.1em $/$%
   \kern-.15em \lower .25ex \hbox{\the\scriptfont0 #2}}%
```

Les macros suivants produisent des logos qui sont utiles dans le monde TEX. Le logo $\mathcal{AMS}$-TEX viens de la page 420 de *The TEXbook* et 999 de la traduction française. Le logo LATEX est légèrement modifié par rapport à celui de `latex.tex` (nous utilisons une police différente pour le 'A') ; de même, le logo BibTEX emploie `\sevenrm` au lieu d'une véritable police de capitale-et- petite-capitale. Le fichier source `.mf` pour le logo METAFONT est donné dans le manuel METAFONT manuel :

Knuth, Donald E.,*The METAFONTbook*. Reading, Mass.: Addison-Wesley, 1986.

```
\def\LaTeX{L\kern-.26em \raise.6ex\hbox{\fiverm A}%
   \kern-.15em TeX}%
\def\AMSTeX{$\cal A\kern-.1667em \lower.5ex\hbox{$\cal M$}%
   \kern-.125em S$-\TeX}%
\def\BibTeX{{\rm B\kern-.05em {\sevenrm I\kern-.025em B}%
   \kern-.08em T\kern-.1667em \lower.7ex\hbox{E}%
   \kern-.125emX}}%
\font\mflogo = logo10
\def\MF{{\mflogo META}{\tenrm \-}{\mflogo FONT}}%
```

Les deux macros suivantes produisent des boîtes. `\blackbox` produit une "bulle carrée", utilisé dans les macros de liste (p. 308). `\makeblankbox` (de page 311 de *The TEXbook* et 999 de la traduction française) produit un rectangle non rempli, avec l'épaisseur des traits de bordure données par les arguments.

```
\def\blackbox{\vrule height .8ex width .6ex depth -.2ex}%
\def\makeblankbox#1#2{%
   \hbox{\lower\dp0\vbox{\hidehrule{#1}{#2}%
      \kern -#1% overlap rules
      \hbox to \wd0{\hidevrule{#1}{#2}%
         \raise\ht0\vbox to #1{}% vrule height
         \lower\dp0\vtop to #1{}% vrule depth
         \hfil\hidevrule{#2}{#1}}%
      \kern-#1\hidehrule{#2}{#1}}}}%
\def\hidehrule#1#2{\kern-#1\hrule height#1 depth#2
```

```
    \kern-#2}%
\def\hidevrule#1#2{\kern-#1{\dimen0 = #1
    \advance\dimen0 by #2 \vrule width\dimen0}\kern-#2}%
```

`\numbername` produit la forme en toute lettre d'un nombre. (si le nombre est plus grands que dix, la macro ne reproduit que les chiffres de son argument.)

```
\def\numbername#1{\ifcase#1%
   zero\or one\or two\or three\or four\or five%
   \or six\or seven\or eight\or nine\or ten\or #1\fi}%
```

`\testfileexistence` détermine si un fichier `\jobname.#1` est non-vide et place `\iffileexists` convenablement. Le nom de fichier dans l'argument n'a pas besoin de finir dans un token d'espace puisque la macro fournit la marque d'espace.

```
\newif\iffileexists
\def\testfileexistence#1{\begingroup
      \immediate\openin0 = \jobname.#1\space
      \ifeof 0\global\fileexistsfalse
      \else \global\fileexiststrue\fi
      \immediate\closein0
   \endgroup}%
```

Affichages

Par défaut, TEX centres le matériel affiché (le matériel entre `$$`). `\leftdisplays` provoque des affichages justifiés à gauche par défaut. Vous pouvez retourner aux affichages centrés avec `\centereddisplays`.

Les macros ici sont plus généralistes qu'elles doivent l'être pour ne faire que des affichages justifiés à gauche. Pour chaque affichage, `\ifeqno` sera vrai si un `\eqno` apparaît dans l'affichage. `\ifleqno` sera vrai si un `\leqno` apparaît. Si l'un ou l'autre sorte d'équation numérotée se apparaît, `\eqn` produit le texte du numéro de l'équation. `\eq` produit toujours le texte de l'équation lui-même.

Ces macros sont basées sur le code de la page 376 de *The TEXbook* et 999 de la traduction française.

```
\newif\ifeqno \newif\ifleqno
\newtoks\@eqtoks \newtoks\@eqnotoks
\def\eq{\the\@eqtoks}\def\eqn{\the\@eqnotoks}%
\def\displaysetup#1$${%
   \@displaytest#1\eqno\eqno\@displaytest}%
\def\@displaytest#1\eqno#2\eqno#3\@displaytest{%
   \if #3% No \eqno, check for \leqno:
      \@ldisplaytest#1\leqno\leqno\@ldisplaytest
```

```
    \else
       \eqnotrue \leqnofalse % Have \eqno, not \leqno.
       \@eqnotoks = {#2}\@eqtoks = {#1}%
    \fi
    \generaldisplay$$}%
\def\@ldisplaytest#1\leqno#2\leqno#3\@ldisplaytest{%
    \@eqtoks = {#1}%
    \if #3%
       \eqnofalse % No \leqno; we're done.
    \else
       \eqnotrue \leqnotrue % Have \leqno.
       \@eqnotoks = {#2}%
    \fi}%
```

Vous pouvez composer des affichages différemment en définissant votre propre macro analogue à `\leftdisplays`. La définition de macro doit placer l'appel à `\displaysetup` dans `\everydisplay` afin de s'assurer que `\displaysetup` soit appelé au début de chaque affichage. La définition de macro doit également inclure une définition de `\generaldisplay`.

```
\newtoks\previouseverydisplay
\def\leftdisplays{%
   \previouseverydisplay = \everydisplay
   \everydisplay =
     {\the\previouseverydisplay \displaysetup}%
   \def\generaldisplay{%
      \leftline{%
         \strut \indent \hskip\leftskip
         \dimen0 = \parindent
         \advance\dimen0 by \leftskip
         \advance\displaywidth by -\dimen0
         \@redefinealignmentdisplays
         \ifeqno \ifleqno
            \kern-\dimen0
            \rlap{$\displaystyle\eqn$}%
            \kern\dimen0
         \fi\fi
         $\displaystyle{\eq}$%
         \ifeqno \ifleqno\else
            \hfill $\displaystyle{\eqn}$%
         \fi\fi}}%
\def\centereddisplays{\let\displaysetup = \relax}%
```

`\leftdisplays` doit travailler dans la douleur pour s'assurer que `\displaylines`, `\eqalignno` et `\leqalignno` fonctionnent toujours correctement. `\eq` est composé en mode mathématique et `\halign` est illégal dans ce mode. Nous utilisons `\vcenter` pour changer le contexte pour que `\halign` redevienne légal. Nous enlevons également les commandes

`\hfil` à gauche du patron pour obtenir la justification à droite. à part ces changements, les macros sont les mêmes que dans `plain.tex`.

```
\def\@redefinealignmentdisplays{%
   \def\displaylines##1{\displ@y
      \vcenter{\halign{\hbox to\displaywidth{$\@lign
            \displaystyle####\hfil$}\crcr##1\crcr}}}%
   \def\eqalignno##1{\displ@y
      \vcenter{\halign to\displaywidth{%
             $\@lign\displaystyle{####}$\tabskip\z@skip
            &$\@lign\displaystyle{{}####}$
               \hfil\tabskip\centering
            &\llap{$\@lign####$}\tabskip\z@skip\crcr
            ##1\crcr}}}%
   \def\leqalignno##1{\displ@y
      \vcenter{\halign to\displaywidth{%
             $\@lign\displaystyle{####}$\tabskip\z@skip
            &$\@lign\displaystyle{{}####}
               $\hfil\tabskip\centering
            &\kern-\displaywidth
             \rlap{\kern-\parindent\kern-\leftskip$
                \@lign####$}%
             \tabskip\displaywidth\crcr
            ##1\crcr}}}%
```

heure du jour

Quand TEX démarre, il détermine les valeurs des paramètres `\time`, `\day`, `\month` et `\year`. `\monthname` produit le nom du mois, abrégé en trois lettres. `\timestring` produit l'heure courante, comme dans "1:14 p.m.". `\timestamp` produit le texte de la date complète, comme dans "23 avr 1964 1:14 p.m.".

```
\def\monthname{%
   \ifcase\month
      \or Jan\or Feb\or Mar\or Apr\or May\or Jun%
      \or Jul\or Aug\or Sep\or Oct\or Nov\or Dec%
   \fi}%
\def\timestring{\begingroup
   \count0 = \time \divide\count0 by 60
   \count2 = \count0   % The hour.
   \count4 = \time \multiply\count0 by 60
   \advance\count4 by -\count0   % The minute.
   \ifnum\count4<10 \toks1 = {0}% Get a leading zero.
   \else            \toks1 = {}%
```

```
    \fi
    \ifnum\count2<12 \toks0 = {a.m.}%
    \else            \toks0 = {p.m.}%
       \advance\count2 by -12
    \fi
    \ifnum\count2=0 \count2 = 12 \fi % Make midnight '12'.
    \number\count2:\the\toks1 \number\count4
    \thinspace \the\toks0
  \endgroup}%
  \def\timestamp{\number\day\space\monthname\space
    \number\year\quad\timestring}%
```

Listes

`\numberedlist` produit des listes numérotées ; `\endnumberedlist` les termine. `\unorderedlist` est analogue. Pour l'une ou l'autre de ces dernières, les items à l'intérieur des listes commencent par `\li` ("list item"). Vous peut mettre `\listcompact` au début d'une liste si vous ne voulez aucun espace additionnel entre les éléments de cette liste. Des listes peuvent être imbriquées arbitrairement.

Vous pouvez contrôler plus généralement l'espacement entre les éléments en assignant des valeurs aux registres énumérés ci-dessous. Si les éléments de vos listes tendent à être longs, vous pourriez vouloir rendre `\interitemskip` différent de zéro. L'indentation gauche de chaque élément de liste est donnée par `\parindent` plus `\listleftindent` ; l'indentation droite de chaque élément de liste est donnée par `\listright% indent`.

```
\newskip\abovelistskip  \abovelistskip = .5\baselineskip
\newskip\interitemskip  \interitemskip = 0pt
\newskip\belowlistskip  \belowlistskip = .5\baselineskip
\newdimen\listleftindent  \listleftindent = \parindent
\newdimen\listrightindent \listrightindent = 0pt
\def\listcompact{\interitemskip = 0pt \relax}%
```

Les listes numérotées et non numérotées utilisent les macros qui suivent. Nous ne changeons pas `\parindent`, puisque beaucoup de macros existantes, comme par exemple, `\footnote`, en dépendent. Nous devons expliquer la possibilité que les éléments fassent plus d'un paragraphe de long. Dans ce cas-ci, tous les paragraphes après le premier sont indentés. Nous utilisons `\leftskip` et `\rightskip` pour indenter les éléments d'une liste. L'indentation des affichages est expliquée par des changements de `\every% display`.

```
\newdimen\@listindent
\def\beginlist{%
```

```
    \@listindent = \parindent
    \advance\@listindent by \listleftindent
    \everydisplay = \expandafter{\the\everydisplay
       % Don't lose user's \everydisplay:
       \advance\displayindent by \@listindent
       \advance\displaywidth by -\@listindent
       \advance\displaywidth by -\listrightindent}%
    \nobreak\vskip\abovelistskip
    \parskip = 0pt
    % \leftskip shifts nested lists to the right on the page.
    \advance\leftskip by \@listindent
    \advance\rightskip by \listrightindent}%
  \def\printitem{\par\noindent
    \llap{\hskip-\listleftindent \marker \enspace}}%
  \def\endlist{\vskip\belowlistskip}%
```

Vous pouvez changer la manière dont les labels d'éléments sont composées en redéfinissant la macro `\numberedmarker`.

```
  \newcount\numberedlistdepth \newcount\itemnumber
  \newcount\itemletter
  \def\numberedmarker{%
     \ifcase\numberedlistdepth
         (impossible)%
     \or \itemnumberout)%
     \or \itemletterout)%
     \else *%
     \fi}%
```

Voici les définitions de `\numberedlist` et de `\unorderedlist`. ces définitions ont la même structure.

```
  \def\numberedlist{\environment{@numbered-list}%
     \advance\numberedlistdepth by 1
     \itemnumber = 1 \itemletter = `a
     \beginlist \let\marker = \numberedmarker
     \def\li{%
        \ifnum\itemnumber=1\else \vskip\interitemskip \fi
        \printitem
        \advance\itemnumber by 1 \advance\itemletter by 1
     }}%
  \def\itemnumberout{\number\itemnumber}%
  \def\itemletterout{\char\itemletter}%
  \def\endnumberedlist{\par
     \endenvironment{@numbered-list}\endlist}%


  \newcount\unorderedlistdepth
  \def\unorderedmarker{%
```

```
    \ifcase\unorderedlistdepth
        (impossible)%
    \or \blackbox
    \or ---%
    \else *%
    \fi}%
\def\unorderedlist{\environment{@unordered-list}%
    \advance\unorderedlistdepth by 1
    \beginlist \itemnumber = 1
    \let\marker = \unorderedmarker
    \def\li{%
       \ifnum\itemnumber=1\else \vskip\interitemskip \fi
       \printitem \advance\itemnumber by 1
    }}%
\def\endunorderedlist{\par
    \endenvironment{@unordered-list}\endlist}%
```

Listing verbatim

La macro `\listing` produit le listing verbatim d'un fichier spécifié en police `\tt`. Elle est basée sur le code de la page 380 de *The TEXbook* et 999 de la traduction française. Les tabulations produisent un montant d'espace fixe et les sauts de pages produisent une coupure de page. D'autres caractères de contrôle produisent ce qui s'avère être à cette position dans la police, qui n'est généralement pas très utile. En redéfinissant `\setuplistinghook`, vous pouvez prendre les mesures complémentaires qui sont appropriées pour vos polices et/ou environnements particuliers avant que le fichier ne soit lu.

```
\def\listing#1{%
   \par \begingroup \@setuplisting \setuplistinghook
   \input #1 \endgroup}%
\let\setuplistinghook = \empty
\def\@setuplisting{%
   \uncatcodespecials
   \obeywhitespace \makeactive\` \makeactive\^^I
   \def^^L{\vfill\eject}\tt}%
{\makeactive\` \gdef`{\relax\lq}}% Defeat ligatures.
{\makeactive\^^I\gdef^^I{\hskip8\fontdimen2\tt \relax}}%
```

Table des matières

La macro `\writetocentry` écrit une macro dans le fichier `\jobname.toc`. Le premier argument de `\writetocentry`, par exemple, "chapitre", est utilisé pour composer le nom de la macro appelée. Le deuxième argument est le texte qui doit apparaître dans la table des matières. `\writetocentry` appose le numéro de la page à l'appel de la macro. Par exemple :

```
\writetocentry{chapter}{Introduction}
```

produira la ligne :

```
\tocchapterentry{Introduction}{2}
```

dans le fichier `.toc`, ce qui indique que 'Introduction' débute en page 2.

Vous pouvez utiliser `\writenumberedtocentry` pour apporter un troisième paramètre, tel qu'un numéro de chapitre. Par exemple :

```
\writenumberedtocentry{chapter}{The second chapter}{2}
```

écrira une ligne :

```
\tocchapterentry{The second chapter}{2}{14}
```

Vous pouvez aussi faire un `\write` vers le `\tocfile` vous-même.

```
\newwrite\tocfile \newif\iftocfileopened
\def\opentocfile{\iftocfileopened\else
      \tocfileopenedtrue
      \immediate\openout\tocfile = \jobname.toc
\fi}%
\def\writenumberedtocentry#1#2#3{\ifrewritetocfile
   \opentocfile
   \write\tocfile{%
      \expandafter\noexpand \csname toc#1entry\endcsname
      {#2}{#3}{\folio}}%
\ignorespaces\fi}%
\def\writenumberedtocentry#1#2#3{\ifrewritetocfile
   \opentocfile
   \write\tocfile{%
      \expandafter\noexpand \csname toc#1entry\endcsname
      {#2}{#3}{\folio}}%
\ignorespaces\fi}%
```

Pour produire une table des matières, lisez le fichier `.toc` avec `\readtocfile`. Vous devez appeler `\readtocfile` avant le premier `\writetocentry`. Quand vous traitez la table des matières sans la régénérer, vous ne devriez pas récrire le fichier `.toc`—si vous le

faites, son contenu sera perdu. La commande `\rewritetocfilefalse` empêchera la réécriture.

```
\newif\ifrewritetocfile \rewritetocfiletrue
\def\readtocfile{\testfileexistence{toc}%
   \iffileexists
      \input \jobname.toc
      \ifrewritetocfile \opentocfile \fi
   \fi}%
```

Voici quelques définitions de macros `\toc...entry` possibles. Ces définitions ne sont indiquées que comme exemple—aligner des traits à travers la ligne n'est normalement pas la meilleure manière de composer une table des matières.

```
\def\tocchapterentry#1#2{\line{\bf #1 \dotfill\ #2}}%
\def\tocsectionentry#1#2{%
   \line{\quad\sl #1 \dotfill\ \rm #2}}%
\def\tocsubsectionentry#1#2{%
   \line{\qquad\rm #1 \dotfill\ #2}}%
```

Références croisées

Les macros qui suivent fournissent des références croisées symbolique, pour que vous puissiez vous référer à quelque chose dans une autre partie d'un document par son nom et non son numéro de page réel. `\xrdef{foo}` définit une étiquette `foo` comme étant le numéro de page courant et `\xrefn{foo}` produit ce numéro de page, par exemple, 77. Plus fréquement, vous voudrez dire quelque chose comme "voir p. 77", ainsi `\xref{foo}` produit ' p. 77". Si `foo` n'est pas défini, un message d'avertissement sera donné. `\xrefwarningfalse` supprime l'avertissement.

Ces macros n'assurent aucune protection contre des définitions dupliquées. Vous pouvez vérifier ces définitions dupliquées en triant le fichier de références croisées et en vérifiant, mécaniquement ou *de visu*, les définitions adjacentes du même symbole.

```
\newif\ifxrefwarning  \xrefwarningtrue
\def\xrdef#1{\begingroup
   \xrlabel{#1}%
   \edef\@wr{\@writexrdef{\the\@xrlabeltoks}}%
   \@wr
   \endgroup \ignorespaces}%
\def\@writexrdef#1{\write\reffile{%
     \string\gdef
        \expandafter\string\csname#1\endcsname
        {\noexpand\folio}\percentchar}}%
```

```
\def\xrefnumber#1{%
   \xrlabel{#1}%
   % \@xrlabeltoks now has the control sequence name.
   \toks0 =
      \expandafter{\csname\the\@xrlabeltoks\endcsname}%
   \expandafter \ifx\the\toks0\relax
      \ifxrefwarning \message{Undefined label
         `\tokstostring{#1}'.}\fi
      {\let\spacesub = \space
       \expandafter\xdef\the\toks0
          {`{\tt \tokstostring{#1}}'}}\fi
   \the\toks0}%
\def\xref#1{p.\thinspace\xrefnumber{#1}}%
\def\xrefn#1{\xrefnumber{#1}}%
```

Cette macro transforme une étiquette en liste de tokens de caractère dans le registre de tokens `\labeltoks`. Une étiquette peut inclure des blancs et des séquences de contrôle aussi bien que les caractères normaux, mais elle ne peut pas inclure d'accolades.

```
\newtoks\@xrlabeltoks
\def\xrlabel#1{\begingroup
      \escapechar = `\_ \edef\tts{\tokstostring{#1_}}%
      \global\@xrlabeltoks = \expandafter{\tts}%
   \endgroup}%
```

Il faut deux passages pour obtenir des références croisées correctes, puisque les définitions sont écrites dans le fichier auxiliaire `\jobname.aux`. `\readreffile` les relit. Si vous ne saisissez pas cette commande avant la première définition, vous perdrez toutes les définitions de l'exécution précédente.

```
\newwrite\reffile \newif\ifreffileopened
\def\openreffile{\ifreffileopened\else
      \reffileopenedtrue
      \immediate\openout\reffile = \jobname.aux
   \fi}%
\def\readreffile{%
   \testfileexistence{aux}%
   \iffileexists
      \begingroup
         \@setletters
         \input \jobname.aux
      \endgroup
   \else
      \message{No cross-reference file; I won't give you
         warnings about undefined labels.}%
      \xrefwarningfalse
   \fi
```

```
   \openreffile}%
\def\@setletters{%
   \catcode`_ = \letter \catcode`+ = \letter
   \catcode`- = \letter \catcode`@ = \letter
   \catcode`0 = \letter \catcode`1 = \letter
   \catcode`2 = \letter \catcode`3 = \letter
   \catcode`4 = \letter \catcode`5 = \letter
   \catcode`6 = \letter \catcode`7 = \letter
   \catcode`8 = \letter \catcode`9 = \letter
   \catcode`( = \letter \catcode`) = \letter}%
```

Vous pouvez donnez des noms symboliques aux équations de la même manière, en utilisant `\eqdef` et `\eqref`. `\eqdef` insère sa propre commande `\eqno`, donc il doit être appelé à un endroit où `\eqno` est légal.

```
\newcount\eqnumber
\def\eqdef#1{\global\advance\eqnumber by 1
   \expandafter\xdef
      \csname#1eqref\endcsname{\the\eqnumber}%
   \immediate\write\reffile{\string\def
      \expandafter\string\csname#1eqref\endcsname
         {\the\eqnumber}}%
   \eqno
   \eqprint{\the\eqnumber}}%
```

`\eqref` produit le "(numéro de l'équation)". Vous pouvez obtenir un formatage plus fantaisiste en redéfinissant `\eqprint`. Par exemple, vous pourriez la redéfinir pour que les numéros d'équation incluent le numéro du chapitre.

```
\def\eqref#1{%
   \expandafter \ifx \csname#1eqref\endcsname \relax
      \ifxrefwarning \message{Undefined equation label
         `#1'.}\fi
      \expandafter\def\csname#1eqref\endcsname{00}%
   \else \eqprint{\csname#1eqref\endcsname}%
   \fi}%
\def\eqprint#1{(#1)}%
```

Environnements

Ces macros vous permettent de définir vos propres groupes appelés (environnements) pour des parties de votre manuscrit. Comme les groupes de TEX, ces groupes peuvent être emboîtés, et en fait leur emboîtement peut être entrelacé avec l'emboîtement des groupes de TEX. Si les noms au début et à la fin d'un environnement ne s'assortissent pas, vous recevrez un message d'erreur. Les macros sont conçues pour que le

message que vous recevez quand une telle erreur se produit vous donne une bonne chance de localiser la cause de l'erreur facilement.

Vous débutez un environnement avec `\environment {foo}` et le finissez avec `\endenvironment{foo}`, où `foo` est le nom de l'environnement. Nos macros améliorent légèrement la réponse à l'exercice 5.7 du *The TEXbook*, en faisant quelques contrôles sur les paires `\begingroup` et `\endgroup`, et s'assurant aussi que les paires `\environment` et `\endenvironment` s'assortissent.

```
\def\environment#1{\ifx\@groupname\undefined\else
      \errhelp = \@unnamedendgrouphelp
      \errmessage{`\@groupname' was not closed by
         \string\endenvironment}\fi
   \def\@groupname{#1}%
   \begingroup
      \let\@groupname = \undefined \ignorespaces}%
\def\endenvironment#1{\endgroup
   \def\@thearg{#1}%
   \ifx\@groupname\@thearg
   \else
      \ifx\@groupname\undefined
         \errhelp = \@isolatedendenvironmenthelp
         \errmessage{Isolated
            \string\endenvironment\space for `#1'}%
      \else
         \errhelp = \@mismatchedenvironmenthelp
         \errmessage{Environment `#1' ended,
            but `\@groupname' started}%
         \endgroup % Probably a typo in the names.
      \fi
   \fi
   \let\@groupname = \undefined \ignorespaces}%
```

Nous définissons également des messages d'aide pour chacune des erreurs ci-dessus.

```
\newhelp\@unnamedendgrouphelp{%
   Most likely, you just forgot an^^J%
   \string\endenvironment.
   Maybe you should try inserting another^^J%
   \string\endgroup to recover.}%
\newhelp\@isolatedendenvironmenthelp{%
   You ended an environment X, but^^J%
   no \string\environment\space to start it
   is anywhere in sight.^^J%
   You might also be at an
   \string\endenvironment\space that would match^^J%
   a \string\begingroup, i.e., you forgot an
```

```
   \string\endgroup.}%
\newhelp\@mismatchedenvironmenthelp{%
   You started an environment X, but^^J%
   you ended it with Y.  Maybe you made a typo
   in one or the other^^J%
   of the names.}%
```

Certains environnements ne doivent pas se produire dans d'autres environnements. Appelons ces derniers "environnements externes [outer]". `\checkenv` vérifie qu'aucun environnement externe n'est actuellement effectif, sinon il se plaint. Pour utiliser `\checkenv`, vous devez mettre la commande `\environmenttrue` au début de chaque environnement externe.

```
\newif\ifenvironment
\def\checkenv{%
   \ifenvironment
      \errhelp = \@interwovenenvhelp
      \errmessage{Interwoven environments}%
      \endgroup
   \fi}%
\newhelp\@interwovenenvhelp{%
   Perhaps you forgot to end the previous^^J%
   environment? I'm finishing off the current group,^^J%
   hoping that will fix it.}%
```

Justification

Les trois macros `\flushleft`, `\flushright` et `\center` justifient le texte des lignes suivantes de la manière indiquée. La commande doit apparaître seule sur une ligne. La commande et le texte doivent être enfermés dans un groupe—la fin du groupe indique la fin du texte. Le groupe entier est placé comme un paragraphe simple, avec des lignes complétées d'un côté ou de l'autres comme indiqué. Les lignes blanches sont reproduites.

```
\begingroup
   \catcode `\^^M = \active
   \globaldefs = 1 %
   \def\flushleft{\beforejustify %
      \aftergroup\@endflushleft %
      \def^^M{\null\hfil\break}%
      \def\@eateol^^M{}\@eateol}%
   \def\flushright{\beforejustify %
      \aftergroup\@endflushright %
      \def^^M{\break\null\hfil}%
```

```
      \def\@eateol^^M{\hfil\null}\@eateol}%
   \def\center {\beforejustify %
      \aftergroup\@endcenter %
      \def^^M{\hfil\break\null\hfil}%
      \def\@eateol^^M{\hfil\null}\@eateol}%
\endgroup
```

Les commandes suivantes sont appelées en raison du `\aftergroup` dans les définitions de `\flushleft`, `\flush right` et `\center`. Elles effectuent les opérations de nettoyage nécessaires .

```
\def\@endflushleft{\unpenalty
   {\parfillskip = 0pt plus 1 fil\par}%
   \ignorespaces}%
\def\@endflushright{%
   % Remove the \hfil\null\break we just put on.
   \unskip \setbox0=\lastbox \unpenalty
   % We have fil glue at the left of the line;
   % \parfillskip shouldn't affect that.
   {\parfillskip = 0pt \par}\ignorespaces}%
\def\@endcenter{%
   % Remove the \hfil\null\break we just put on.
   \unskip \setbox0=\lastbox \unpenalty
   % We have fil glue at the left of the line;
   % \parfillskip must balance it.
   {\parfillskip = 0pt plus 1fil \par}\ignorespaces}%
\def\beforejustify{%
   \par\noindent
   \catcode`\^^M = \active
   \checkenv \environmenttrue}%
```

Tables

La macro `\makecolumns` vous permet de donner toutes les entrées d'une table sans vous inquiéter de l'endroit où les colonnes se coupent. Par exemple, si vous saisissez une longue liste alphabétique qui sera composée en plusieurs colonnes, vous ne saurez habituellement pas à l'avance où une colonne finit et où la suivante commence. Par ailleurs, si un autre élément est ajouté, les coupures de colonne changeront.

`\makecolumns` prend deux arguments (délimités) : le nombre total de rangées et le nombre de colonnes de la table. Par exemple, '`\makecolumns 37/3:`' désigne une table de trois colonnes dont les rangées sont les 37 prochaines lignes. Vous pouvez ajuster le positionnement de la table sur la page en changeant `\parindent`, qui détermine l'espace à gauche et `\hsize`, qui détermine l'espace entre la marge gauche de la page et la

droite du bloc. Vous pouvez permettre une coupure de page au-dessus du `\valign` en changeant `\abovecolumnspenalty`..

```
\newcount\abovecolumnspenalty
\abovecolumnspenalty = 10000
\newcount\@linestogo      % Lines remaining to process.
\newcount\@linestogoincolumn % Lines remaining in column.
\newcount\@columndepth    % Number of lines in a column.
\newdimen\@columnwidth    % Width of each column.
\newtoks\crtok  \crtok = {\cr}%
\def\makecolumns#1/#2: {\par \begingroup
   \@columndepth = #1 \advance\@columndepth by #2
   \advance\@columndepth by -1
   \divide \@columndepth by #2
   \@linestogoincolumn = \@columndepth \@linestogo = #1
   \def\@endcolumnactions{%
      \ifnum \@linestogo<2
         \the\crtok \egroup \endgroup \par
            % End \valign and \makecolumns.
      \else
         \global\advance\@linestogo by -1
         \ifnum\@linestogoincolumn<2
            \global\@linestogoincolumn = \@columndepth
            \the\crtok
         \else &\global\advance\@linestogoincolumn by -1
         \fi
      \fi}%
   \makeactive\^^M\letreturn\@endcolumnactions
   \@columnwidth = \hsize
   \advance\@columnwidth by -\parindent
   \divide\@columnwidth by #2
   \penalty\abovecolumnspenalty
   \noindent % It's not a paragraph (usually).
   \valign\bgroup
      &\hbox to \@columnwidth{\strut ##\hfil}\cr
}% The next end-of-line starts everything going.
```

Notes de pied de page

Des notes de bas de page sont composées le plus généralement en utilisant un numéro incrémenté comme marque de référence. Nous définissons la macro `\numberedfootnote` pour faire cela. Il redéfinit également `\vfootnote` pour permettre un formatage légèrement plus générique des notes de bas de page que ne le fait plain TEXLe registre de dimension `\footnotemarkseparation` contrôle l'espace entre la marque

de la notes de bas de page (par exemple, le numéro) et le début du texte. Les tokens `\everyfootnote` sont insérés avant de produire la notes de bas de page.

Les définitions plain TEX de `\footnote` et `\vfootnote` sont préservés en `\@plainfootnote` et `\@plainvfootnote` au cas où vous auriez besoin d'elles.

```
\newcount\footnotenumber \newtoks\everyfootnote
\newdimen\footnotemarkseparation
\footnotemarkseparation = .5em
\let\@plainfootnote = \footnote
\let\@plainvfootnote = \vfootnote
\def\vfootnote#1{\insert\footins\bgroup
  \interlinepenalty\interfootnotelinepenalty
  \splittopskip\ht\strutbox \splitmaxdepth\dp\strutbox
  \floatingpenalty\@MM
  \leftskip\z@skip \rightskip\z@skip \spaceskip\z@skip
  \xspaceskip\z@skip
  \everypar = {}%
  \the\everyfootnote
  \indent\llap{#1\kern\footnotemarkseparation}\footstrut
  \futurelet\next\fo@t}%
\def\numberedfootnote{\global\advance\footnotenumber by 1
  \@plainfootnote{$^{\number\footnotenumber}$}}%
```

Double colonnes

La commande `\doublecolumns` commence la composition en double colonnes, alors que la commande `\singlecolumn` remet la composition en colonne simple. Vous pouvez commuter dans les deux sens entre elles sur la même page. Le ressort spécifié par `\abovedoublecolumn% skip` et `\belowdoublecolumnskip` est insérée avant.et.après la composition en double colonnes.

L'approche est dérivée de la page 417 de *The TEXbook* et 999 de la traduction française.

```
\newskip\abovedoublecolumnskip
\newskip\belowdoublecolumnskip
\abovedoublecolumnskip = \bigskipamount
\belowdoublecolumnskip = \bigskipamount
\newdimen\gutter            \gutter = 2pc
\newdimen\doublecolumnhsize \doublecolumnhsize = \hsize
\newbox\@partialpage        \newdimen\singlecolumnhsize
\newdimen\singlecolumnvsize \newtoks\previousoutput
\def\doublecolumns{\par % Don't start in horizontal mode.
```

```
\previousoutput = \expandafter{\the\output}
\advance\doublecolumnhsize by -\gutter
\divide\doublecolumnhsize by 2
\output = {\global\setbox\@partialpage =
      \vbox{\unvbox255\vskip\abovedoublecolumnskip}}%
\pagegoal = \pagetotal \break % Expands \output above.
\output = {\doublecolumnoutput}%
\singlecolumnhsize = \hsize
\singlecolumnvsize = \vsize
\hsize = \doublecolumnhsize \vsize = 2\vsize}%
```

Le macro `\@doublecolumnsplit` fait le dédoublement réel. On suppose que le source est insérée en colonne simple. Si vous ne voulez pas que cela soit le cas, vous devrez modifier la routine de résultat. Après que `\@doublecolumnsplit` ait effectué son travail, `\box255` aura le matériel en double colonnes. Le matériel en double colonnes sera précédé par n'importe quel matériel en colonne simple qui aura été composé avant que `\doublecolumns` ait été appelé. `\box4` aura le matériel qui ne s'est pas adapté à la page.

```
\def\@doublecolumnsplit{%
   \splittopskip = \topskip \splitmaxdepth = \maxdepth
   \dimen0 = \singlecolumnvsize
      \advance\dimen0 by -\ht\@partialpage
      \advance\dimen0 by -\ht\footins
      \advance\dimen0 by -\skip\footins
      \advance\dimen0 by -\ht\topins
   \begingroup
      \vbadness = 10000
      \global\setbox1=\vsplit255 to \dimen0 \wd1=\hsize
      \global\setbox3=\vsplit255 to \dimen0 \wd3=\hsize
   \endgroup
   \global\setbox4=\vbox{\unvbox255
      \penalty\outputpenalty}%
   \global\setbox255=\vbox{\unvbox\@partialpage
      \hbox to \singlecolumnhsize{\box1\hfil\box3}%
      \vfill}}%
```

`\doublecolumnoutput` est la routine de sortie réelle. Nous appelons l'ancienne `\output` pour faire le travail de sortie de la boîte.

```
\def\doublecolumnoutput{\@doublecolumnsplit
   \hsize = \singlecolumnhsize \vsize = \singlecolumnvsize
   \previousoutput \unvbox4}%
```

`\singlecolumn` reprend la composition en une colonne. Il suppose que `\doublecolumns` a été appelé.

```
\def\singlecolumn{\par % Don't start in horizontal mode.
   \output = {\global\setbox1 =
```

```
   \vbox{\unvbox255\vskip\abovedoublecolumnskip}}%
\pagegoal = \pagetotal \break \setbox255 = \box1
{\singlecolumnvsize = \ht255
   \divide\singlecolumnvsize by 2
   \advance\singlecolumnvsize by +\ht\@partialpage
   \advance\singlecolumnvsize by +\ht\footins
   \advance\singlecolumnvsize by +\skip\footins
   \advance\singlecolumnvsize by +\ht\topins
\@doublecolumnsplit}%
\hsize = \singlecolumnhsize
\vsize = \singlecolumnvsize
\output = \expandafter{\the\previousoutput}%
\unvbox255}%
```

Terminer

Nous devons maintenant défaire les changements que nous avons faits quand nous avons commencé (voir la p. 302). Nous donnons également un numéro de version, qui sera plus tard disponible dans `\fmtname` et `\fmtversion`.

```
\let\wlog = \@plainwlog \catcode`@ = \other
\def\fmtname{eplain}%
{\edef\plainversion{\fmtversion}%
 \xdef\fmtversion{1.0: 15 May 1990
   (and plain \plainversion)}%
}%
```

13 Sommaire des commandes

Cette section contient des descriptions sur une ligne des commandes primitives deTEX et des commandes TEX définies dans plain TEX. Cela inclus les séquences de contrôle et les caractères spéciaux. Nous avons omis les commandes qui ne sont utilisées qu'a l'usage interne de la définition plain TEX (Annexe B de *The TEXbook* et de la traduction française). Notez que les caractères ordinaires comme '`a`' ou '`6`' sont aussi des commandes, et même les plus communes (voir "caractère", p. 53).

Pour rendre la desciption la plus brève possible, nous avons adopté certaines conventions :

- Un astérisque devant une commande indique que la commande est une primitive, c'est-à-dire, construite dans le programme TEX (voir "primitive", p. 90).
- Les mots "musique", "ponctuation", "fonction", "symbole", "relation", "délimiteur", or "opérateur" dans une description de commande implique que la commande n'est légale qu'en mode mathématique.
- Le verbe "afficher" s'applique à l'information que TEX envoie au fichier `.log`, sauf indication contraire. Si `\tracingonline` est positif, TEX envoie également cette information sur le terminal. Nous utilisons le nom "affichage" pour nous référer aux affichages mathématiques (voir p. 79), c'est-à-dire, à ce qui se trouve entre des `$$`.
- La phrase "produit x" indique que la commande composera x et mettra le résultat dans une boîte. Nous omettons parfois "produit" quand l'omission ne peut porter à confusion. Par exemple, nous décrivons `\alpha` comme "lettre Grecque mathématique α" et non "produit la lettre grecque mathématique α".

`*\␣` espace inter mot (p. 110)
`\!` espace fin négatif en mathématique (p. 222)

\" accent tréma en texte, comme dans ö (p. 106)
introduit un paramètre de macro ou indique où le texte source va dans un préambule d'alignement (p. 73, p. 46)
\# produit le caractère # de la police courante (p. 104)
$ commence ou fini une formule mathématique (p. 16)
\$ produit le caractère $ de la police courante (p. 104)
* % commence un commentaire (p. 13)
\% produit le caractère % de la police courante (p. 104)
& sépare les modèles et les entrées dans un alignement (p. 184)
\& produit le caractère & de la police courante (p. 104)
' symbole prime en mathématique, comme dans p' (p. 196)
\' accent aigu en texte, comme dans é (p. 106)
* symbole multiplication qui autorise une coupure de ligne (p. 198)
\+ début d'une ligne tabulée (p. 181)
\, espace fin en mathématique (p. 222)
*\- spécifie un point de césure légal (p. 132)
\. accent point en texte, comme dans ṅ (p. 106)
*\/ correction italique pour le caractère précédent (p. 112)
\; espace épaissi en mathématique (p. 222)
\= accent macron en texte, comme dans r̄ (p. 106)
* \ débute une séquence de contrôle (p. 10)
\> espace moyen en mathématique (p. 222)
^ produit une sous-formule spécifiée en exposant (p. 205)
\^ accent circonflexe en texte, comme dans ô (p. 106)
^^L équivalent à la primitive \par (p. 116)
*^^M une fin de ligne (p. 111)
_ produit une sous-formule spécifiée en indice (p. 205)
_ soulignement : _ (p. 104)
\` accent grave en texte, comme dans è (p. 106)
{ débute un groupe (p. 235)
\{ délimiteur accolade ouvrante en mathématique : { (p. 199)
\| lignes parallèles en mathématique : ‖ (p. 196)
} fin d'un groupe (p. 235)
\} délimiteur accolade fermante en mathématique : } (p. 199)
~ espace inter-mot sur lequel une ligne ne sera pas coupée (p. 111)
\~ accent tilde en texte, comme dans ã (p. 106)
\aa lettre scandinave : å (p. 103)
\AA lettre scandinave : Å (p. 103)
*\above produit une fraction avec une barre d'épaisseur spécifiée (p. 208)
*\abovedisplayshortskip ressort TEX inséré avant un affichage quand la ligne précédente dépasse l'indentation d'affichage, par défaut 0 pt plus 3 pt (p. 225)

*\abovedisplayskip ressort TEX inséré avant un affichage quand la ligne précédente ne dépasse pas l'indentation d'affichage, par default 12 pt plus 3 pt minus 9 pt (p. 225)
*\abovewithdelims produit une fraction avec une barre d'épaisseur spécifiée et entourée des délimiteurs spécifiés (p. 209)
*\accent met l'accent spécifié sur le caractère suivant (p. 106)
\active code de catégorie pour des caractères actifs, c'est-à-dire, le nombre 13 (p. 259)
\acute accent aigu en mathématique, comme dans $\acute{x}$ (p. 207)
*\adjdemerits démérites additionnels pour une coupure de ligne qui seront mises dans des lignes adjacentes avec un espacement de mot incompatible, par défaut 10000 (p. 131)
*\advance ajoute un nombre à un registre \count (p. 253)
\advancepageno si \pageno est positif, ajoute un ; s'il est négatif, soustrait un (p. 148)
\ae ligature æ (p. 103)
\AE ligature Æ (p. 103)
*\afterassignment attend, pour développer le token suivant, que l'assignement d'après soit effectué (p. 237)
*\aftergroup attend, pour développer le token suivant, la fin du groupe courant (p. 237)
\aleph lettre hébraïque seule en mathématique : $\aleph$ (p. 196)
\allowbreak fait \penalty0, c'est-à-dire, autorise une coupure de ligne ou de page là où elle ne devrait pas être (p. 127, p. 142)
\alpha lettre mathématique grecque α (p. 195)
\amalg opérateur d'amalgamation : $\amalg$ (p. 197)
\angle symbole d'angle : $\angle$ (p. 196)
\approx relation d'approximation : $\approx$ (p. 198)
\arccos fonction arc cosinus : $\arccos$ (p. 201)
\arcsin fonction arc sinus : $\arcsin$ (p. 201)
\arctan fonction arc tangente : $\arctan$ (p. 201)
\arg fonction argument (phase) : $\arg$ (p. 201)
\arrowvert partie verticale d'une double flèche extensible (p. 220)
\Arrowvert partie verticale d'une flèche simple extensible (p. 220)
\ast opérateur astérisque : $\ast$ (p. 197)
\asymp relation asymptote : $\asymp$ (p. 198)
*\atop produit une fraction sans barre de fraction (p. 208)
*\atopwithdelims produit une fraction sans barre de fraction et entourée des délimiteurs spécifiés (p. 209)
\b accent barre du dessous en mathématique, comme dans $\underline{x}$ (p. 207)
\backslash symbole antislash : $\backslash$ (p. 196)
*\badness la médiocrité du ressort mis dans la dernière boîte fabriquée (p. 176)
\bar accent barre en mathématique, comme dans $\bar{x}$ (p. 207)

*\baselineskip ressort pour la distance verticale normale entre une ligne de base et la suivante, par défaut 12 pt (p. 139)
*\batchmode ne s'arrête pas sur les erreurs et n'écrit rien sur le terminal (p. 261)
*\begingroup débute un groupe terminé par \endgroup (p. 235)
\beginsection débute une subdivision majeur d'un document (p. 135)
*\belowdisplayshortskip ressort que TEX insère après un affichage quand la ligne précédente dépasse l'indentation de l'affichage, par défaut 7 pt plus 0.3 pt minus 4 pt (p. 225)
*\belowdisplayskip ressort que TEX insère après un affichage quand la ligne précédente ne dépasse pas l'indentation de l'affichage, par défaut 12 pt plus 3 pt minus 9 pt (p. 225)
\beta lettre grecque mathématique β (p. 195)
\bf utilise le gras, c'est-à-dire, fait \tenbf\fam=\bffam (p. 109)
\bffam famille grasse en mathématique (p. 218)
\bgroup caractère de début de groupe implicite (p. 235)
\big rend le délimiteur spécifié plus grand qu'à l'ordinaire, mais encore assez petit pour du texte (p. 219)
\Big rend le délimiteur spécifié haut de 11.5 pt (p. 219)
\bigbreak indique une coupure de page désirable avec \penalty-200 et produit un ressort \bigskipamount (p. 143)
\bigcap grand opérateur cap (non, cela ne produit pas une grande lettre capitale !) : $\bigcap$ (p. 202)
\bigcirc grand opérateur cercle : $\bigcirc$ (p. 197)
\bigcup grand opérateur coupe : $\bigcup$ (p. 202)
\bigg rend le délimiteur spécifié haut de 14.5 pt (p. 219)
\Bigg rend le délimiteur spécifié haut de 17.5 pt (p. 219)
\biggl taille comme \bigg, mais espacé comme une ouverture (p. 219)
\Biggl taille comme \Bigg, mais espacé comme une ouverture (p. 219)
\biggm taille comme \bigg, mais espacé comme une relation (p. 219)
\Biggm taille comme \Bigg, mais espacé comme une relation (p. 219)
\biggr taille comme \bigg, mais espacé comme une fermeture (p. 219)
\Biggr taille comme \Bigg, mais espacé comme une fermeture (p. 219)
\bigl taille comme \big, mais espacé comme une ouverture (p. 219)
\Bigl taille comme \Big, mais espacé comme une ouverture (p. 219)
\bigm taille comme \big, mais espacé comme une relation (p. 219)
\Bigm taille comme \Big, mais espacé comme une relation (p. 219)
\bigodot grand opérateur cercle pointé : $\bigodot$ (p. 202)
\bigoplus grand opérateur cercle plus : $\bigoplus$ (p. 202)
\bigotimes grand opérateur cercle multiplié : $\bigotimes$ (p. 202)
\bigr taille comme \big, mais espacé comme une fermeture (p. 219)
\Bigr taille comme \Big, mais espacé comme une fermeture (p. 219)
\bigskip produit un ressort \bigskipamount (p. 160)

\bigskipamount ressort pour une grand saut vertical, par défaut 12 pt plus 4 pt minus 4 pt (p. 161)
\bigsqcup grand opérateur en coupe carré : $\bigsqcup$ (p. 202)
\bigtriangledown opérateur en triangle pointant vers le bas : $\bigtriangledown$ (p. 197)
\bigtriangleup opérateur en triangle pointant vers le haut : $\bigtriangleup$ (p. 197)
\biguplus grand opérateur en coupe avec plus : $\biguplus$ (p. 202)
\bigvee grand opérateur logique "ou" : $\bigvee$ (p. 202)
\bigwedge grand opérateur logique "et" : $\bigwedge$ (p. 202)
*\binoppenalty pénalité supplémentaire pour une coupure après un opérateur mathématique binaire, par défaut 700 (p. 132)
\bmod opérateur modulus, comme dans $n \bmod 2$ (p. 201)
\bordermatrix produit une matrice avec des labels de rangés et de colonnes (p. 213)
\bot lattice bottom symbol : $\bot$ (p. 196)
*\botmark le dernier élément marqué de la page qui vient d'être mise en boîte (p. 150)
\bowtie bowtie relation : $\bowtie$ (p. 198)
*\box affiche la boîte d'un registre de boîte spécifié dans la liste courante, et vide le registre (p. 170)
*\boxmaxdepth profondeur maximale des vbox, par défaut \maxdimen (p. 169)
\brace $n\brace k$ produit braced notation : $\binom{n}{k}$ (p. 208)
\bracevert vertical portion of extensible large brace (p. 220)
\brack $n\brack k$ produces bracketed notation : $\binom{n}{k}$ (p. 208)
\break fait \penalty-10000, c'est-à-dire., force une coupure de ligne ou de page (p. 126, p. 142)
\breve accent bref en mathématique, comme dans $\breve{x}$ (p. 207)
*\brokenpenalty pénalité pour une coupure de ligne sur un élément discrètionnaire, par défaut 100 (p. 145)
\buildrel produit une formule spécifié sur la relation spécifié (p. 210)
\bullet bullet operation : $\bullet$ (p. 197)
\bye \vfill la dernière page avec de l'espace blanc, la \supereject, et \end l'exécution (p. 255)
\c accent cédille en texte, comme dans ç (p. 106)
\cal utilise une police calligraphique pour des lettres capitales en mathématique, comme dans $\mathcal{XYZ}$ (p. 217)
\cap cap operator : $\cap$ (p. 197)
\cases produit des cas en mathématique, comme dans $\begin{cases} \cdots \\ \cdots \end{cases}$ (p. 209)
*\catcode le code de catégorie d'un caractère spécifié (p. 259)
\cdot opérateur point centré : $\cdot$ (p. 197)
\cdotp ponctuation point centré : $\cdot$ (p. 204)
\cdots points centrés en mathématique : $\cdots$ (p. 211)

\centerline produit une ligne avec son texte centré (p. 115)
*\char produit le caractère de la police courante avec le code spécifié (p. 105)
*\chardef défini une séquence de contrôle spécifié comme étant un code de caractère, un nombre entre 0 et 255 (p. 240)
\check accent tchèque en mathématique, comme dans $\check{x}$ (p. 207)
\chi lettre grecque mathématique χ (p. 195)
\choose $n\choose k$ produit une notation combinatoire : $\binom{n}{k}$ (p. 208)
\circ opération cercle : $\circ$ (p. 197)
*\cleaders produce leaders with half of leftover space before the first box, and half after the last (p. 179)
\cleartabs efface toutes les tabulations pour des alignements tabulés (p. 184)
*\closein ferme un flot d'entrée spécifié (p. 256)
*\closeout ferme un flot de sortie spécifié (p. 257)
*\clubpenalty pénalité additionnelle pour une ligne seule restante avant une coupure de page, par défaut 150 (p. 144)
\clubsuit symbole trèfle : $\clubsuit$ (p. 196)
\colon colon punctation symbol en mathématique : $:$ (p. 204)
\cong relation congruence : $\cong$ (p. 198)
\coprod opérateur co-produit : $\coprod$ (p. 202)
*\copy comme \box, mais n'efface pas le registre (p. 170)
\copyright marque de copyright : © (p. 104)
\cos fonction cosinus : $\cos$ (p. 201)
\cosh fonction cosinus hyperbolique : $\cosh$ (p. 201)
\cot fonction cotangente : $\cot$ (p. 201)
\coth fonction cotangente hyperbolique : $\coth$ (p. 201)
*\count le registre entier spécifié (p. 250)
*\countdef défini une séquence de contrôle spécifiée comme étant un nombre correspondant à un registre \count (p. 253)
*\cr termine une rangé (ou une colonne) dans un alignement (p. 186)
*\crcr ne fait rien si la dernière commande était \cr ou \noalign ; autrement, équivalent à \cr (p. 187)
\csc fonction co-sécante : $\csc$ (p. 201)
*\csname débute un nom de séquence de contrôle devant être terminé par \endcsname (p. 241)
\cup cup operator : $\cup$ (p. 197)
\d accent point en dessous en texte, comme dans $\underset{.}{r}$ (p. 106)
\dag symbole dague en texte : † (p. 104)
\dagger opérateur dague en mathématique : $\dagger$ (p. 197)
\dashv right turnstile relation : $\dashv$ (p. 198)
*\day jour courant du mois, comme un nombre (p. 233)
\ddag symbole double dague en texte : ‡ (p. 104)
\ddagger opérateur double dague en mathématique : $\ddagger$ (p. 197)

\ddot accent deux points en mathématique : $\ddot{x}$ (p. 207)
\ddots points en diagonale en mathématique : $\ddots$ (p. 211)
*\deadcycles nombre d'initialisations d'\output depuis le dernier \shipout (p. 155)
*\def défini une séquence de contrôle comme étant une macro (p. 238)
*\defaulthyphenchar code du caractère de césure par défaut (p. 135)
*\defaultskewchar default accent skewing character code (p. 221)
\deg fonction degré : deg (p. 201)
*\delcode le code délimiteur d'un caractère spécifié (p. 260)
*\delimiter produit un délimiteur spécifié (p. 213)
*\delimiterfactor 1000 fois le ratio de la taille minimum d'un délimiteur par rapport à la taille qui recouvrirait complètement la formule, par défaut 901 (p. 213)
*\delimitershortfall différence minimum entre une hauteur de formule et une hauteur de délimiteur, par défaut 5 pt (p. 213)
\delta lettre grecque mathématique δ (p. 195)
\Delta lettre grecque mathématique Δ (p. 195)
\det fonction déterminant : det (p. 201)
\diamond opérateur diamant : $\diamond$ (p. 197)
\diamondsuit symbole carré : $\diamondsuit$ (p. 196)
\dim fonction dimension : dim (p. 201)
*\dimen le registre de dimension spécifié (p. 250)
*\dimendef défini une séquence de contrôle spécifiée comme étant un nombre correspondant à un registre \dimen (p. 253)
*\discretionary spécifie trois textes, les deux premiers pour avant et après une coupure de ligne, le troisième quand il n'y a pas de coupure de ligne (p. 132)
*\displayindent TEX met ceci pour l'indentation d'un affichage (p. 224)
*\displaylimits place des limites sur et sous des opérateurs seulement en styles d'affichage (p. 203)
\displaylines produit un affichage multi-lignes spécifié avec chaque ligne centrée (p. 216)
*\displaystyle utilise la taille displaystyle dans une formule (p. 206)
*\displaywidowpenalty pénalité pour une ligne seule débutant une page juste avant un affichage, par défaut 50 (p. 144)
*\displaywidth TEX met ceci pour la largeur d'une affichage (p. 224)
\div opérateur division : $\div$ (p. 197)
*\divide divise un registre \count spécifié par un entier spécifié (p. 254)
\dot accent point en mathématique, comme dans $\dot{x}$ (p. 207)
\doteq relation d'égalité pointée : $\doteq$ (p. 198)
\dotfill rempli un espace horizontal délimité avec des points (p. 180)
\dots ellipse pour des suites : $x_1, \ldots, x_n$ (p. 105)
*\doublehyphendemerits démérites pour deux lignes consécutives se terminant par de césures, par défaut 10000 (p. 131)

\downarrow relation : ↓ (p. 200)

\Downarrow relation : ⇓ (p. 200)

\downbracefill fill enclosing hbox with a downwards facing brace : ⏟ (p. 220)

*\dp la profondeur de la boîte d'un registre de boîte spécifié (p. 173)

*\dump termine l'exécution et produit un fichier de format (p. 271)

*\edef défini une séquence de contrôle comme étant une macro, développant immédiatement le texte de remplacement (p. 238)

\egroup caractère de fin de groupe implicite (p. 235)

\eject termine le paragraphe courant et force une coupure de page, en étirant la page courante (p. 143)

\ell script letter en mathématique : ℓ (p. 196)

*\else cas faux ou par défaut pour une condition (p. 248)

*\emergencystretch étirement supplémentaire ajouté à toute ligne si \tolerance n'est pas satisfait (p. 129)

\empty macro dont le développement ne fait rien (p. 250)

\emptyset symbole d'ensemble vide : ∅ (p. 196)

*\end \output la dernière page et termine l'exécution (p. 255)

*\endcsname termine un nom de séquence de contrôle commencé par \csname (p. 241)

\endgraf équivalent à la primitive \par (p. 117)

*\endgroup termine un groupé débuté par \begingroup (p. 235)

*\endinput termine l'entrée du fichier courant (p. 255)

\endinsert fin d'insertion (p. 153)

\endline équivalent à la primitive \cr (p. 187)

*\endlinechar caractère que TEX insère à la fin de chaque ligne entrée, par défaut ^^M (p. 260)

\enskip ressort horizontal de largeur 1/2 em (p. 160)

\enspace crénage de 1/2 em (p. 160)

\epsilon lettre grecque mathématique ϵ (p. 195)

\eqalign produit un affichage multi-lignes spécifié dont les parties indiqués sont alignées verticalement (p. 216)

\eqalignno produit un affichage multi-lignes spécifié avec des numéros d'équation dont les parties indiquées sont alignées verticalement (p. 216)

*\eqno met un numéro d'équation spécifié à droite d'un affichage (p. 215)

\equiv relation d'équivalence : ≡ (p. 198)

*\errhelp liste de token dont le développement TEX s'affiche quand l'utilisateur demande de l'aide en réponse à un \errmessage (p. 270)

*\errmessage donne un message d'erreur spécifié (p. 269)

*\errorcontextlines le nombre de lignes de contexte que TEX affiche pour une erreur, par défaut 5 (p. 270)

*\errorstopmode stoppe pour une interaction sur des messages d'erreur (p. 260)
*\escapechar caractère avec lequel TEX précède les noms de séquence de contrôle qui sont affichés (p. 234)
\eta lettre grecque mathématique η (p. 195)
*\everycr liste de token que TEX développe après un \cr ou un \crcr non suivi de \cr ou de \noalign (p. 192)
*\everydisplay liste de token que TEX développe quand un affichage mathématique débute (p. 226)
*\everyhbox liste de token que TEX développe quand une hbox débute (p. 170)
*\everyjob liste de token que TEX développe quand une exécution débute (p. 271)
*\everymath liste de token que TEX développe quand un texte en mode mathématique débute (p. 226)
*\everypar liste de token que TEX développe quand une paragraphe débute (p. 119)
*\everyvbox liste de token que TEX développe quand une vbox débute (p. 170)
*\exhyphenpenalty pénalité supplémentaire pour une coupure de ligne après une césure explicite, par défaut 50 (p. 131)
\exists symbole "il existe" : $\exists$ (p. 196)
\exp fonction exponentielle : exp (p. 201)
*\expandafter ne développe le token suivant qu'après développement du token le suivant (p. 241)
*\fam famille de police que TEX utilise pour des caractères de classe sept (c'est-à-dire, des variables) en math (p. 218)
*\fi termine une condition (p. 248)
\filbreak force une coupure de page à moins que le texte contienne un autre \filbreak convienne aussi sur la page (p. 143)
*\finalhyphendemerits pénalité pour l'avant-dernière ligne coupée sur une césure, par défaut 5000 (p. 132)
*\firstmark premier élément marqué sur la page qui vient d'être mise en boîte (p. 150)
\fivebf utilise une police grasse en 5 points, cmbx5 (p. 108)
\fivei utilise une police mathématique italique en 5 points, cmmi5 (p. 108)
\fiverm utilise une police romaine en 5 points, cmr5 (p. 108)
\fivesy utilise une police de symbole en 5 points, cmsy5 (p. 108)
\flat symbole bémol en musique : $\flat$ (p. 196)
*\floatingpenalty pénalité pour des insertions qui sont séparées sur des pages, par défaut 0 (p. 145)
\fmtname nom du format courant (p. 233)
\fmtversion numéro de version du format courant (p. 233)
\folio produit \pageno comme des caractères ; en chiffres romains s'il est négatif (p. 149)

*\font défini une séquence de contrôle spécifique pour sélectionner une police (p. 229)
*\fontdimen un paramètre spécifié d'une police spécifiée (p. 230)
*\fontname produit le nom de fichier d'une police spécifiée en caractères (p. 235)
\footline liste de token qui produit une ligne en bas de chaque page (p. 149)
\footnote produit une note de pied de page spécifiée avec une marque de référence spécifiée (p. 151)
\forall symbole "pour tout" : ∀ (p. 196)
\frenchspacing rend l'espacement inter-mot indépendant de la ponctuation (p. 112)
\frown relation frown : ⌢ (p. 198)
*\futurelet assigne le troisième token suivant à une séquence de contrôle spécifiée, puis développe le deuxième token (p. 240)
\gamma lettre grecque mathématique γ (p. 195)
\Gamma lettre grecque mathématique Γ (p. 195)
\gcd fonction plus grand dénominateur commun : gcd (p. 201)
*\gdef équivalent à \global\def, c'est-à-dire, défini une macro globale (p. 239)
\ge relation plus grand ou égal : ≥ (p. 198)
\geq équivalent à \ge (p. 198)
\gets gets relation : ← (p. 200)
\gg relation beaucoup plus grand que : ≫ (p. 198)
*\global rend la définition suivante globale (p. 236)
*\globaldefs overrides \global prefixes on assignments (p. 236)
\goodbreak indique une coupure de page souhaitable avec \penalty-500 (p. 143)
\grave accent grave en mathématique, comme dans $\grave{x}$ (p. 207)
\H accent tréma hongrois en texte, comme dans ő (p. 106)
*\halign aligne du texte en colonnes (p. 184)
\hang indente le paragraphe courant de \parindent (p. 123)
*\hangafter numéro de ligne débutant une indentation hanging (p. 123)
*\hangindent espace pour indentation hanging (p. 123)
\hat accent circonflexe en mathématique, comme dans $\hat{x}$ (p. 207)
*\hbadness badness threshold for reporting underfull or overfull hboxes, par défaut 1000 (p. 176)
\hbar symbole mathématique : ℏ (p. 196)
*\hbox produit une hbox spécifiée (p. 166)
\headline liste de token qui produit la ligne en haut de chaque page (p. 149)
\heartsuit symbole cœur : ♡ (p. 196)
*\hfil produit un ressort horizontal infiniment étirable (p. 163)

*\hfill produit un ressort horizontal encore plus infiniment étirable que celui produit par \hfil (p. 163)
*\hfilneg produit un ressort horizontal négatif infiniment étirable (p. 165)
*\hfuzz space threshold for reporting overfull hboxes, par défaut 0.1 pt (p. 176)
\hglue produit un ressort horizontal qui ne disparait pas sur des coupure de ligne (p. 162)
\hidewidth ignore la largeur d'une entrée d'un alignement, so that it extends out from its box in the direction of the \hidewidth (p. 191)
*\hoffset page offset relative to one inch from the paper's left edge (p. 146)
*\holdinginserts si positive, n'enlève pas d'insertions de la page courante (p. 155)
\hom fontion homologie : hom (p. 201)
\hookleftarrow relation : ↩ (p. 200)
\hookrightarrow relation : ↪ (p. 200)
\hphantom produit une formule invisible de hauteur et largeur zéro mais de largeur naturelle (p. 175)
*\hrule produit un filet horizontal ; légale seulement en modes verticaux (p. 178)
\hrulefill rempli un espace délimité d'un filet horizontal (p. 180)
*\hsize longeur de ligne, par défaut 6.5 in (p. 120)
*\hskip produit un ressort horizontal spécifié (p. 161)
*\hss produit un ressort horizontal qui est infiniment étirable et infiniment rétrécissable (p. 164)
*\ht la hauteur de la boîte d'un registre de boîte spécifié (p. 173)
*\hyphenation ajoute des mots spécifiés au dictionnaire d'exception de césure (p. 133)
*\hyphenchar le caractère de césure d'une police spécifiée (p. 134)
*\hyphenpenalty pénalité supplémentaire pour une coupure de ligne sur une césure, par défaut 50 (p. 131)
\i lettre 'ı' sans point à utiliser avec des accents (p. 106)
\ialign débute un \halign avec le ressort \tabskip à zéro et \everycr vide (p. 186)
*\if teste si deux tokens spécifiés ont le même code de caractère (p. 243)
*\ifcase développe un cas n pour un valeur n spécifiée (p. 247)
*\ifcat teste si deux tokens spécifiés ont le même code de catégorie (p. 243)
*\ifdim teste for a specified relationship between two specified dimensions (p. 245)
*\ifeof test la fin d'un fichier spécifié (p. 247)
\iff relation si et seulement si : ⟺ (p. 200)
*\iffalse test qui est toujours faux (p. 247)
*\ifhbox teste si un registre de boîte spécifié contient une hbox (p. 246)

*\ifhmode teste si TEX est dans un mode horizontal (p. 246)
*\ifinner teste si TEX est dans un mode interne (p. 246)
*\ifmmode teste si TEX est dans un mode mathématique (p. 246)
*\ifnum test for a specified relationship between two specified numbers (p. 245)
*\ifodd teste si un nombre spécifié est impair (p. 245)
*\iftrue test qui est toujours vrai (p. 247)
*\ifvbox teste si un registre de boîte spécifié contient une vbox (p. 246)
*\ifvmode teste si TEX est dans un mode vertical (p. 246)
*\ifvoid teste si un registre de boîte spécifié est vide (p. 246)
*\ifx teste si deux tokens sont les même, ou si deux macros ont les mêmes définitions finales (p. 244)
*\ignorespaces ignore tous les tokens espaces suivants (p. 260)
\Im symbole d'une partie imaginaire d'un complexe : $\Im$ (p. 196)
\imath lettre '$\imath$' sans point à utiliser avec des accents mathématiques (p. 196)
*\immediate effectue l'opération de fichier spécifiée sans délai (p. 258)
\in containment relation : $\in$ (p. 198)
*\indent produit une boîte vide de largeur \parindent et entre en mode horizontal (p. 117)
\inf fonction inférieur : inf (p. 201)
\infty symbole infini : ∞ (p. 196)
*\input commence à lire à partir d'un fichier spécifié (p. 255)
*\inputlineno le numéro de ligne courante du fichier d'entrée courant (p. 256)
*\insert produit une insertion d'un classe spécifiée (p. 153)
*\insertpenalties somme des pénalités des insertions (p. 145)
\int symbole intégrale : $\int$ (p. 202)
*\interlinepenalty pénalité supplémentaire pour une coupure de page entre des lignes d'un paragraphe, par défaut 0 (p. 144)
\iota lettre grecque mathématique ι (p. 195)
\it utilise des italiques, c'est-à-dire, fait \tenit\fam=\itfam (p. 109)
\item débute un paragraphe avec une indentation accrochée de \parindent et précédée d'un label spécifié (p. 136)
\itemitem comme \item, mais avec une indentation de 2\parindent (p. 136)
\itfam famille italique en mathématique (p. 218)
\j lettre 'ȷ' sans point, pour utiliser avec des accents (p. 106)
\jmath lettre '$\jmath$' sans point, pour utiliser avec des accents mathématiques (p. 196)
*\jobname nom de base du fichier par lequel TEX a été appelé (p. 233)
\jot unité de mesure for opening up displays (p. 223)
\kappa lettre grecque mathématique κ (p. 195)
\ker fonction kern : ker (p. 201)

*\kern produit un montant d'espace spécifié sur lequel une coupure n'est pas autorisée (p. 163)
\l lettre polonaise : ł (p. 103)
\L lettre polonaise : Ł (p. 103)
\lambda lettre grecque mathématique λ (p. 195)
\Lambda lettre grecque mathématique Λ (p. 195)
\land opérateur logique "et" : $\wedge$ (p. 197)
\langle left angle delimiter : $\langle$ (p. 199)
*\language le jeu de patrons de césure courant (p. 134)
*\lastbox retrouve et enlève le dernier élément de la liste courante, si c'est une boîte (p. 177)
*\lastkern retrouve le dernier élément de la liste courante, si c'est un crénage (p. 177)
*\lastpenalty retrieve the last item from the current list, if it's a penalty (p. 177)
*\lastskip retrieve the last item from the current list, if it's glue (p. 177)
\lbrace délimiteur accolade gauche : { (p. 199)
\lbrack délimiteur crochet gauche : [(p. 199)
*\lccode le code de caractère pour la forme minuscule d'une lettre (p. 109)
\lceil délimiteur plafond gauche : $\lceil$ (p. 199)
\ldotp point sur la ligne de base comme ponctuation : . (p. 204)
\ldots point sur la ligne de base en mathématique : ... (p. 211)
\le relation inférieur ou égal : $\leq$ (p. 198)
*\leaders rempli un espace horizontal ou vertical spécifié en répétant un filet ou une boîte spécifiée (p. 179)
*\left produit le délimiteur spécifié, en l'agrandissant pour couvrir la sous-formule suivante se finissant par \right (p. 212)
\leftarrow relation : $\leftarrow$ (p. 200)
\Leftarrow relation : $\Leftarrow$ (p. 200)
\leftarrowfill rempli la hbox contenante avec un \leftarrow : $\longleftarrow$ (p. 181)
\leftharpoondown relation : $\leftharpoondown$ (p. 200)
\leftharpoonup relation : $\leftharpoonup$ (p. 200)
*\lefthyphenmin taille du plus petit fragment de mot que TEX autorise avant une césure en début d'un mot, par défaut 2 (p. 134)
\leftline produit une ligne avec son texte poussé vers la marge de gauche (p. 115)
\leftrightarrow relation : $\leftrightarrow$ (p. 200)
\Leftrightarrow relation : $\Leftrightarrow$ (p. 200)
*\leftskip ressort que TEX insère à gauche de chaque ligne (p. 121)
\leq équivalent à \le (p. 198)

`\leqalignno` produit un affichage multi-ligne spécifié avec des numéros d'équation à gauche dont les parties indiquées sont alignées verticalement (p. 216)

`*\leqno` dépose un numéro d'équation spécifié à gauche d'un affichage (p. 215)

`*\let` défini une séquence de contrôle comme étant le token suivant (p. 240)

`\lfloor` délimiteur plancher gauche : $\lfloor$ (p. 199)

`\lg` fonction logarithme : lg (p. 201)

`\lgroup` délimiteur groupe gauche (la plus petite taille est montrée ici) : $\left($ (p. 212)

`\lim` fonction limite : lim (p. 201)

`\liminf` fonction limite inférieur : $\liminf$ (p. 201)

`*\limits` place un exposant sur et un indice sous un grand opérateur (p. 203)

`\limsup` fonction limite supérieur : $\limsup$ (p. 201)

`\line` produit une ligne de caractère justifiée (p. 115)

`*\linepenalty` pénalité pour coupure de ligne ajoutée à chaque ligne, par défaut 10 (p. 130)

`*\lineskip` ressort vertical d'une ligne de base à la suivante si les lines sont plus proche l'une de l'autre que `\lineskiplimit`, par défaut 1 pt (p. 139)

`*\lineskiplimit` threshold for using `\lineskip` instead of `\baselineskip`, par défaut 0 pt (p. 139)

`\ll` relation beaucoup moins que : $\ll$ (p. 198)

`\llap` produit du texte (sans largeur) à partir de la gauche de la position courante (p. 115)

`\lmoustache` moitié haute d'une grande accolade : $\int$ (p. 220)

`\ln` fonction logarithme naturel : ln (p. 201)

`\lnot` symbole logique "non" : $\neg$ (p. 196)

`\log` fonction logarithme : log (p. 201)

`*\long` autorise des tokens `\par` dans le(s) argument(s) de la définition suivante (p. 239)

`\longleftarrow` relation : $\longleftarrow$ (p. 200)

`\Longleftarrow` relation : $\Longleftarrow$ (p. 200)

`\longleftrightarrow` relation : $\longleftrightarrow$ (p. 200)

`\Longleftrightarrow` relation : $\Longleftrightarrow$ (p. 200)

`\longmapsto` relation : $\longmapsto$ (p. 200)

`\longrightarrow` relation : $\longrightarrow$ (p. 200)

`\Longrightarrow` relation : $\Longrightarrow$ (p. 200)

`\loop` débute une boucle devant se finir par `\repeat` (p. 248)

`*\looseness` différence entre le nombre de lignes d'un paragraphe que vous voulez par rapport au nombre optimal (p. 130)

\lor opérateur logique "ou" : $\vee$ (p. 197)
*\lower abaisse une boîte spécifié d'un montant spécifié (p. 172)
*\lowercase converti les lettres capitales du texte spécifié en minuscule (p. 110)
\lq caractère quote gauche pour du texte : ‘ (p. 104)
*\mag 1000 fois le ratio pour agrandir toutes les dimensions (p. 231)
\magnification comme \mag, mais n'agrandit pas la taille de la page (p. 231)
\magstep $1000 \cdot 1.2^n$ pour un n spécifié (p. 232)
\magstephalf $1000 \cdot \sqrt{1.2}$ (p. 232)
\mapsto relation : $\mapsto$ (p. 200)
*\mark produit un élément marqué avec un texte spécifié (p. 150)
*\mathaccent met un accent mathématique spécifié sur le caractère suivant (p. 207)
*\mathbin espace une sous-formule spécifiée comme une opération binaire (p. 226)
*\mathchar produit le caractère mathématique de mathcode spécifié (p. 105)
*\mathchardef défini une séquence de contrôle spécifié comme étant un mathcode, un nombre entre 0 et $2^{15} - 1$ (p. 240)
*\mathchoice sélectionne une des quatre sous-formules mathématiques spécifiées selon le style courant (p. 206)
*\mathclose espace une sous-formule spécifiée comme un délimiteur fermant (p. 226)
*\mathcode le mathcode d'un caractère spécifié (p. 259)
*\mathinner espace une sous-formule spécifiée comme une formule interne, c'est-à-dire, une fraction (p. 226)
*\mathop espace une sous-formule spécifiée comme un grand opérateur mathématique (p. 226)
*\mathopen espace une sous-formule spécifiée comme un délimiteur ouvrant (p. 226)
*\mathord espace une sous-formule spécifiée comme un caractère ordinaire (p. 226)
\mathpalette produce a \mathchoice which expands a specified control sequence depending on the current style (p. 206)
*\mathpunct espace une sous-formule spécifiée comme une ponctuation (p. 226)
*\mathrel espace une sous-formule spécifiée comme une relation (p. 226)
\mathstrut produit une boîte invisible avec la hauteur et la largeur d'une parenthèse gauche et sans largeur (p. 174)
*\mathsurround space TEX kerns before and after math in text (p. 225)
\matrix produit une matrice spécifiée (p. 213)
\max fonction maximum : max (p. 201)
*\maxdeadcycles valeur de \deadcycles sur laquelle TEX se plaint, et utilise alors sa propre routine de sortie, par défaut 25 (p. 155)

*\maxdepth profondeur maximum de la boîte en bas d'une page, par défaut 4 pt (p. 147)

\maxdimen plus grande dimension acceptable par TEX (p. 252)

*\meaning produit la signification compréhensible humainement d'un token spécifié comme des caractères (p. 234)

\medbreak indique une coupure de page souhaitable avec \penalty-100 et produit un ressort \medskipamount (p. 143)

*\medmuskip ressort pour un espace mathématique moyen, par défaut 4 mu plus 2 mu minus 4 mu (p. 222)

\medskip produit un ressort \medskipamount (p. 160)

\medskipamount ressort pour un saut vertical moyen, par défaut 6 ptplus 2 pt minus 2 pt (p. 161)

*\message montre le développement du texte spécifié sur le terminal (p. 269)

\mid middle relation : $\mid$ (p. 198)

\midinsert produit le texte spécifié à la position courante si possible, sinon en haut de la page suivante (p. 152)

\min fonction minimum : min (p. 201)

\mit utilise des italiques mathématiques, c'est-à-dire, fait \fam=1 (p. 217)

*\mkern produit un crénage spécifié en unités mu en mathématique (p. 223)

\models models relation : $\models$ (p. 198)

*\month mois courant, comme un nombre (p. 233)

*\moveleft déplace une boîte spécifié à gauche d'un espace spécifié ; légal uniquement en modes verticaux (p. 172)

*\moveright déplace une boîte spécifié à droite d'un espace spécifié ; légal uniquement en modes verticaux (p. 172)

\mp opérateur plus et moins : $\mp$ (p. 197)

*\mskip produit un ressort spécifié en unités mu en mathématique (p. 223)

\mu lettre grecque mathématique μ (p. 195)

*\multiply multiplie un registre \count spécifié par un entier spécifié (p. 254)

\multispan fait traverser l'entrée d'alignement suivante un nombre spécifié de colonnes (ou de rangées) (p. 188)

*\muskip le registre muglue spécifié (p. 250)

*\muskipdef défini une séquence de contrôle spécifié comme un nombre correspondant à un regisstre \muskip (p. 253)

\nabla backwards difference symbol : ∇ (p. 196)

\narrower make both left and right margins narrower by \parindent (p. 121)

\natural symbole naturel en musique : $\natural$ (p. 196)

\nearrow relation flèche montante vers la droite : $\nearrow$ (p. 200)

\ne relation non égal : $\neq$ (p. 198)

\neg symbole logique "non" : ¬ (p. 196)
\negthinspace kern $-1/6$ em (p. 159)
\neq relation non égal : ≠ (p. 198)
\newbox réserve et nomme un registre \box (p. 252)
\newcount réserve et nomme un registre \count (p. 252)
\newdimen réserve et nomme un registre \dimen (p. 252)
\newfam réserve et nomme une famille mathématique (p. 252)
\newhelp nomme un message d'erreur spécifié (p. 270)
\newif défini un nouveau test conditionel avec le nom spécifié (p. 248)
\newinsert nomme une classe d'insertion et réserve des registres \box, \count, \dimen et \skip correspondants (p. 252)
\newlanguage réserve et nomme un \language (p. 252)
*\newlinechar caractère fin de ligne pour \write, etc. (p. 259)
\newmuskip réserve et nomme un registre \muskip (p. 252)
\newread réserve et nomme un flot d'entrée (p. 252)
\newskip réserve et nomme un registre \skip (p. 252)
\newtoks réserve et nomme un registre \toks (p. 252)
\newwrite réserve et nomme un flot de sortie (p. 252)
\ni relation "dans inversé" : ∋ (p. 198)
*\noalign insère du matériel entre des rangées (ou des colonnes) d'un alignement (p. 189)
*\noboundary inhibe des ligatures ou crénages dues au boundarychar de la police courante (p. 107)
\nobreak fait \penalty10000, c'est-à-dire, inhibe une coupure de ligne ou de page (p. 127, p. 142)
*\noexpand supprime le développement du token suivant (p. 242)
*\noindent entre en mode horizontal sans indentet le paragraphe (p. 118)
\nointerlineskip inhibe le ressort inter-ligne avant la ligne suivante (p. 141)
*\nolimits place un exposant et un indice après de grands opérateurs (p. 203)
\nonfrenchspacing rend l'espacement inter-mot dépendant de la ponctuation (p. 112)
*\nonscript inhibe tout ressort ou crénage suivant en styles script et scriptscript (p. 223)
*\nonstopmode ne stoppe pas sur des erreurs, même celles relatives aux fichier manquants (p. 261)
\nopagenumbers inhibe l'impression des numéros de page, c'est-à-dire, fait \footline = \hfil (p. 148)
\normalbaselines met \baselineskip, \lineskip et \lineskiplimit aux valeurs normales pour la taille de la police courante (p. 140)
\normalbaselineskip valeur de \baselineskip pour la taille de la police courante (p. 140)

`\normalbottom` rend la marge du bas identiques de page en page (p. 144)
`\normallineskip` valeur de `\lineskip` pour la taille de caractère courante (p. 140)
`\normallineskiplimit` valeur de `\lineskiplimit` pour la taille de caractère courante (p. 140)
`\not` un slash sans largeur pour construire des négations de relation matématiques, comme dans $\neq$ (p. 199)
`\notin` relation n'appartient pas : $\notin$ (p. 198)
`\nu` lettre grecque mathématique ν (p. 195)
`\null` se développe en une boîte vide (p. 175)
`*\nulldelimiterspace` espace produit par un délimiteur nul, par défaut 1.2 pt (p. 225)
`*\nullfont` police primitive sans characters (p. 108)
`*\number` produit un chiffre spécifié en caractères (p. 232)
`\nwarrow` relation flèche en haut vers la gauche : $\nwarrow$ (p. 200)
`\o` lettre danoise : ø (p. 103)
`\O` lettre danoise : Ø (p. 103)
`\obeylines` rend chaque fin de ligne du fichier entrée équivalente à `\par` (p. 128)
`\obeyspaces` produit un espace dans la sortie pour chaque caractère espace dans l'entrée (p. 113)
`\odot` opération point centré : $\odot$ (p. 197)
`\oe` ligature œ (p. 103)
`\OE` ligature Œ (p. 103)
`\offinterlineskip` inhibe le ressort inter-ligne à partir de maintenant (p. 141)
`\oint` opérateur contour intégral : $\oint$ (p. 202)
`\oldstyle` utilise des chiffres elzéviriens : 1234567890 (p. 217)
`\omega` lettre grecque mathématique ω (p. 195)
`\Omega` lettre grecque mathématique Ω (p. 195)
`\ominus` opérateur moins dans un cercle : $\ominus$ (p. 197)
`*\omit` saute un patron de colonne (ou de rangée) dans un alignement (p. 187)
`*\openin` prépare un flot d'entrée spécifié pour lire un fichier (p. 256)
`*\openout` prépare un flot de sortie spécifié pour écrire dans un fichier (p. 257)
`\openup` augmente `\baselineskip`, `\lineskip` et `\lineskiplimit` d'un montant spécifié (p. 141)
`\oplus` opérateur plus dans un cercle : $\oplus$ (p. 197)
`*\or` sépare les cas d'un `\ifcase` (p. 247)
`\oslash` opérateur divise dans un cercle : $\oslash$ (p. 197)
`\otimes` opérateur multiplie dans un cercle : $\otimes$ (p. 197)
`*\outer` rend la définition de macro suivante illégale dans des contextes dans lesquels des tokens sont absorbé à haute vitesse (p. 239)

*\output liste de token que TEX développe quand il trouve une coupure de page (p. 154)
*\outputpenalty si la coupure de page arrive sur une pénalité, la valeur de cette pénalité ; sinon zéro (p. 155)
*\over produit une fraction avec une barre d'épaisseur par défaut (p. 208)
\overbrace produit une accolade recouvrant le haut d'une formule, comme dans $\overbrace{h+w}$ (p. 210)
*\overfullrule largeur de la réglure d'un "overfull box" (p. 175)
\overleftarrow produit une flèche vers la gauche recouvrant le haut d'une formule, comme dans $\overleftarrow{r+a}$ (p. 210)
*\overline produit une ligne recouvrant le haut d'une formule, comme dans$\overline{2b}$ (p. 210)
\overrightarrow produit une flèche vers la droite recouvrant le haut d'une formule, comme dans $\overrightarrow{i+t}$ (p. 210)
*\overwithdelims produit une fraction avec une barre d'épaisseur par défaut et surmontée des délimiteurs spécifiés (p. 209)
\owns relation appartient : $\ni$ (p. 198)
\P caractère paragraphe en texte : ¶ (p. 104)
*\pagedepth TEX met ceci à la profondeur courante de la page courante (p. 145)
*\pagefilllstretch TEX met ceci au montant d'étirement filll sur la page courante (p. 146)
*\pagefillstretch TEX met ceci au montant d'étirement fill sur la page courante (p. 146)
*\pagefilstretch TEX met ceci au montant d'étirement fil sur la page courante (p. 146)
*\pagegoal TEX met ceci à la hauteur désirée pour la page courante (c'est-à-dire, \vsize quand la première boîte est mise sur la page) (p. 145)
\pageinsert produit le texte spécifié sur la page suivante et utilise jusqu'a la page entière (p. 152)
\pageno le registre \count0, qui contient le numéro de page (éventuellement négatif) (p. 148)
*\pageshrink TEX met ceci au montant total de rétrécissement sur la page courante (p. 146)
*\pagestretch TEX met ceci au montant total de étirement sur la page courante (p. 146)
*\pagetotal TEX met ceci à la hauteur naturelle de la page courante (p. 145)
*\par fini un paragraphe et termine le mode horizontal (p. 116)
\parallel relation parallèle : $\parallel$ (p. 198)
*\parfillskip ressort horizontal que TEX insère à la fin d'un paragraphe (p. 117)
*\parindent espace horizontal que TEX insère au début d'un paragraphe (p. 119)

*\parshape spécifie la largeur et la longueur de chaque ligne dans le prochain paragraphe (p. 124)
*\parskip ressort vertical que TEX insère avant un paragraphe (p. 147)
\partial symbole dérivée partielle : ∂ (p. 196)
*\pausing si positif, stoppe après avoir lu chaque ligne d'entrée pour un remplacement possible (p. 261)
*\penalty produit une pénalité (ou un bonus, si négatif) pour couper une ligne ou une page ici (p. 127, p. 142)
\perp relation perpendiculaire : $\perp$ (p. 198)
\phantom produit une formule invisible avec les dimensions d'une sous formule spécifié (p. 174)
\phi lettre grecque mathématique ϕ (p. 195)
\Phi lettre grecque mathématique Φ (p. 195)
\pi lettre grecque mathématique π (p. 195)
\Pi lettre grecque mathématique Π (p. 195)
\plainoutput routine \output de plain TEX (p. 154)
\pm opérateur plus et moins : $\pm$ (p. 197)
\pmatrix produit une matrice entre parenthèses (p. 213)
\pmod parenthesized modulus notation to put at the end of a formula, as in $x \equiv y + 1 \pmod 2$ (p. 202)
*\postdisplaypenalty pénalité supplémentaire pour une coupure de ligne juste après un affichage, par défaut 0 (p. 144)
\Pr fonction probabilité : Pr (p. 201)
\prec relation précède : $\prec$ (p. 198)
\preceq relation précède ou égale : $\preceq$ (p. 198)
*\predisplaypenalty pénalité supplémentaire pour une coupure de ligne juste avant un affichage, par défaut 0 (p. 144)
*\predisplaysize TEX met ceci à la largeur de la ligne précedant un affichage (p. 224)
*\pretolerance tolérance de médiocrité pour une coupure de ligne sans césure, par défaut 100 (p. 128)
*\prevdepth profondeur de la dernière boîte dans la liste verticale courante (sauf filet) (p. 140)
*\prevgraf TEX met ceci au nombre de ligne dans le paragraphe en plus (en mode horizontal) ou dans le paragraphe précédent (en mode vertical) (p. 126)
\prime symbole prime mathématique, comme dans r' (p. 196)
\proclaim débute un théorème, un lemme, une hypothèse, ... (p. 136)
\prod grand opérateur produit : $\prod$ (p. 202)
\propto relation proportionnel à : $\propto$ (p. 198)
\psi lettre grecque mathématique ψ (p. 195)
\Psi lettre grecque mathématique Ψ (p. 195)
\qquad produit un ressort horizontal de largeur 2 em (p. 160)
\quad produit un ressort horizontal de largeur 1 em (p. 160)

*\radical produit un symbole radical spécifié (p. 215)
\raggedbottom autorise la marge du bas à varier d'une page à l'autre (p. 144)
\raggedright autorise la marge droite à varier d'une line à l'autre (p. 122)
*\raise abaisse une boîte spécifiée d'un montant spécifié (p. 172)
\rangle délimiteur angle fermant : ⟩ (p. 199)
\rbrace délimiteur accolade fermante : } (p. 199)
\rbrack délimiteur crochet fermant :] (p. 199)
\rceil délimiteur plancher fermant : ⌉ (p. 199)
\Re symbole partie réelle de complexe : ℜ (p. 196)
*\read lit une ligne d'un flot d'entrée spécifié (p. 256)
*\relax ne fait rien (p. 249)
*\relpenalty pénalité supplémentaire pour une coupure après une relation, par défaut 500 (p. 132)
\repeat termine une boucle débuté avec \loop (p. 248)
\rfloor délimiteur plafond fermant : ⌋ (p. 199)
\rgroup délimiteur de groupe fermant (la plus petite taille est montrée ici) : ⎭ (p. 212)
\rho lettre grecque mathématique ρ (p. 195)
*\right produiy le délimiteur spécifié à l'extrèmité d'une sous-formule débutée par \left (p. 212)
\rightarrow relation : → (p. 200)
\Rightarrow relation : ⇒ (p. 200)
\rightarrowfill remplit une hbox contenante avec un \rightarrow : ⟶ (p. 181)
\rightharpoondown relation : ⇁ (p. 200)
\rightharpoonup relation : ⇀ (p. 200)
\rightleftharpoons relation : ⇌ (p. 200)
\rightline produit une ligne avec son texte poussé vers la marge droite (p. 115)
*\rightskip ressort que TEX insère à droite de chaque ligne (p. 121)
*\righthyphenmin taille du plus petit fragment de mot que TEX autorise après une césure à la fin d'un mot, par défaut 3 (p. 134)
\rlap produit du texte (sans largeur) à partir de la droite de la position courante (p. 115)
\rm utilise des caractère romain, c'est-à-dire, fait \tenrm\fam=0 (p. 109)
\rmoustache moitié basse d'une grande accolade : ⎱ (p. 220)
\romannumeral produit la représentation en chiffre romain minuscule d'un nombre spécifié, en caractères (p. 232)
\root produit une racine spécifiée d'une sous-formule spécifiée, comme dans $\sqrt[3]{2}$ (p. 215)
\rq caractère quote fermante en texte : ' (p. 104)

\S caractère section en texte : § (p. 104)
\sb caractère indice implicite (p. 205)
*\scriptfont la police de style script dans une famille mathématique spécifiée (p. 219)
*\scriptscriptfont la police de style scriptscript dans une famille mathématique spécifiée (p. 219)
*\scriptscriptstyle utilise la taille de style scriptscript dans une formule (p. 206)
*\scriptspace espace supplémentaire que TEX crène après un exposant ou un indice, par défaut 0.5 pt (p. 226)
*\scriptstyle utilise la taille de style script dans une formule (p. 206)
*\scrollmode ne stoppe pas sur les erreurs, mais stoppe sur des erreurs de fichiers manquants (p. 261)
\searrow relation flèche en bas à gauche : ↘ (p. 200)
\sec fonction sécante : sec (p. 201)
*\setbox défini un registre de boîte spécifié comme étant une boîte (p. 170)
*\setlanguage change une jeu de règles de césure spécifié, mais ne change pas \language (p. 134)
\setminus set difference operator : \ (p. 197)
\settabs défini les tabulations pour une alignement tabulé (p. 182)
\sevenbf utilise une police grasse de 7 points, cmbx7 (p. 108)
\seveni utilise une police mathématique italique de 7 points, cmmi5 (p. 108)
\sevenrm utilise une police romaine de 7 points, cmr7 (p. 108)
\sevensy utilise une police symbole de 7 points, cmsy7 (p. 108)
*\sfcode le code du facteur d'espace d'un caractère spécifié (p. 113)
\sharp symbole dièse en musique : ♯ (p. 196)
*\shipout envoie une boîte vers le fishier .dvi (p. 154)
*\show montre, dans la log et sur le terminal, la signification d'un token spécifié (p. 261)
*\showbox affiche le contenu d'une registre de boîte spécifié (p. 261)
*\showboxbreadth nombre maximum d'éléments montré sur chaque niveau imbriqué, par défaut 5 (p. 269)
*\showboxdepth maximum de niveau imbriqué montré, par défaut 3 (p. 269)
\showhyphens montre, dans la log et sur le terminal, les césure dans un texte spécifié (p. 133)
*\showlists affiche toutes les listes sur lequel il travaille (p. 261)
*\showthe montre, dans la log et sur le terminal, ce que \the produira (p. 261)
\sigma lettre grecque mathématique σ (p. 195)
\Sigma lettre grecque mathématique Σ (p. 195)
\sim relation similarité : $\sim$ (p. 198)
\simeq relation similaire ou egal : $\simeq$ (p. 198)

\sin fonction sinus : sin (p. 201)
\sinh fonction sinus hyperbolique : sinh (p. 201)
\skew déplace un accent spécifié d'un montant spécifié sur un caractère accentué spécifié (p. 220)
*\skewchar caractère d'une police spécifiée utilisé pour positionner des accents (p. 221)
*\skip le registre de ressort spécifié (p. 250)
*\skipdef défini une séquence de contrôle spécifié comme étant un nombre correspondant à un registre \skip (p. 253)
\sl utilise des caractères penchés, c'est-à-dire, fait \tensl\fam=\slfam (p. 109)
\slash caractère / autorisant une coupure de ligne (p. 128)
\slfam famille penchée en mathématique (p. 218)
\smallbreak indique une coupure de page assez désirable avec \penalty-50 et produit un ressort \smallskipamount (p. 143)
\smallint petit symbole d'intégrale : $\smallint$ (p. 202)
\smallskip produit un ressort \smallskipamount (p. 160)
\smallskipamount ressort pour une petit saut vertical, par défaut 3 pt plus 1 pt minus 1 pt (p. 161)
\smash produit une formule sans hauteur, ni profondeur (p. 175)
\smile relation smile : $\smile$ (p. 198)
\sp caractère d'exposant implicite (p. 205)
\space produit un ressort inter-mot normal (p. 111)
*\spacefactor modifie l'étirement et le rétrécissement des ressorts inter-mot s'il différent de 1000 (p. 113)
*\spaceskip si différent de zéro et \spacefactor < 2000, surcharge le ressort inter-mot normal (p. 113)
\spadesuit symbole trèfle : $\spadesuit$ (p. 196)
*\span soit combine des entrées dans un corps d'alignement body soit développe des tokens dans un préambule (p. 188)
*\special écrit des tokens dans le fichier .dvi devant être interprétés par un programme de lecture de DVI (p. 258)
*\splitbotmark dernier élément de marque dans une boîte résultant de \vsplit (p. 150)
*\splitfirstmark premier élément de marque dans une boîte résultant de \vsplit (p. 150)
*\splitmaxdepth profondeur maximum d'une boîte résultant de \vsplit (p. 156)
*\splittopskip ressort que TEX insère en haut d'une boîte résultant de \vsplit (p. 156)
\sqcap opérateur square cap : $\sqcap$ (p. 197)
\sqcup opérateur square cup : $\sqcup$ (p. 197)
\sqrt produit une racine carrée d'une sous-formule, comme dans $\sqrt{2}$ (p. 214)
\sqsubseteq relation square subset ou egal : $\sqsubseteq$ (p. 198)

\sqsupseteq relation square superset ou egal : ⊒ (p. 198)
\ss lettre allemande : ß (p. 103)
\star opérateur étoile : ⋆ (p. 197)
*\string produit un token spécifié, le plus souvent une séquence de contrôle, comme des caractères (p. 234)
\strut boîte sans largeur, mais de hauteur et profondeur d'une ligne standard, de ligne de base à ligne de base, dans la police courante (p. 173)
\subset relation subset : ⊂ (p. 198)
\subseteq relation subset ou egal : ⊆ (p. 198)
\succ relation successeur : ≻ (p. 198)
\succeq relation successeur ou egal : ⪰ (p. 198)
\sum grand opérateur de somme : ∑ (p. 202)
\sup fonction supérieur : sup (p. 201)
\supereject force une coupure de page, et décharge toutes les insertions (p. 143)
\supset relation superset : ⊃ (p. 198)
\supseteq relation superset ou égal : ⊇ (p. 198)
\surd symbole surd : √ (p. 196)
\swarrow relation flèche en bas à gauche : ↙ (p. 200)
\t accent tie-after en texte, comme dans ûu (p. 106)
\tabalign équivalent à \+, sauf s'il n'est pas \outer (p. 181)
*\tabskip ressort entre colonnes (ou rangées) d'un alignement (p. 190)
\tan fonction tangente : tan (p. 201)
\tanh fonction tangente hyperbolique : tanh (p. 201)
\tau lettre grecque mathématique τ (p. 195)
\tenbf utilise une police grasse de 10 points, cmbx10 (p. 108)
\tenex utilise une police d'extension mathématique de 10 points, cmex10 (p. 108)
\teni utilise une police italique mathématique de 10 points, cmmi10 (p. 108)
\tenit utilise une police italique de texte de 10 points, cmti10 (p. 108)
\tenrm utilise une police romaine de texte de 10 points, cmr10 (p. 108)
\tensl utilise une police romaine penchée de 10 points, cmsl10 (p. 108)
\tensy utilise une police de symbole mathématique de 10 points, cmsy10 (p. 108)
\tentt utilise une police de machine à écrire de 10 points, cmtt10 (p. 108)
\TeX produit le logo TEX (p. 104)
*\textfont la police de style dans une famille mathématique spécifiée (p. 219)
\textindent comme \item, mais ne fait pas d'indentation accrochée (p. 118)
*\textstyle utilise une taille textstyle dans une formule (p. 206)

*\the donne la valeur d'un token spécifié (p. 242)
\theta lettre grecque mathématique θ (p. 195)
\Theta lettre grecque mathématique Θ (p. 195)
*\thickmuskip ressort pour une espace mathématique grasse, par défaut 5 mu plus 5 mu (p. 222)
*\thinmuskip ressort pour une espace mathématique fine, par défaut 3 mu (p. 222)
\thinspace crénage de $1/6$ em (p. 159)
\tilde accent tilde en mathématique, comme dans $\tilde{x}$ (p. 207)
*\time l'heure d'aujourd'hui, en minutes depuis minuit (p. 233)
\times opérateur multiplié : $\times$ (p. 197)
*\toks le registre de token spécifié (p. 250)
*\toksdef défini une séquence de contrôle spécifié comme étant un nombre correspondant à un registre \toks (p. 253)
*\tolerance tolérance de médiocrité pour des coupure de ligne avec césure (p. 128)
\to relation mapping : $\to$ (p. 200)
\top symbole lattice top : $\top$ (p. 196)
\topglue produit un ressort vertical spécifié en haut d'une page (p. 162)
\topinsert produit le texte spécifié en haut d'une page (p. 152)
*\topmark \botmark avant que la page courante soit en boîte (p. 150)
*\topskip ressort entre la ligne de tête et la première ligne de texte sur une page, par défaut 10 pt (p. 147)
\tracingall déclenche des traces maximal (p. 269)
*\tracingcommands affiche l'exécution des commandes (p. 264)
*\tracinglostchars affiche les caractères demandés, mais non définis (p. 265)
*\tracingmacros affiche les développements de macros (p. 266)
*\tracingonline montre les sorties de diagnostique sur le terminal aussi bien que dans le fichier log (p. 264)
*\tracingoutput affiche les contenus des boîtes sorties (p. 266)
*\tracingpages affiche les calculs de coupure de page (p. 267)
*\tracingparagraphs affiche les calculs de coupure de ligne (p. 267)
*\tracingrestores affiche les valeurs restaurées à la fin d'un groupe (p. 268)
*\tracingstats affiche les statistiques d'utilisation de la mémoire (p. 268)
\triangle symbole triangle : $\triangle$ (p. 196)
\triangleleft opérateur triangle gauche : $\triangleleft$ (p. 197)
\triangleright opérateur triangle droite : $\triangleright$ (p. 197)
\tt utilise des caractère de machine à écrire, c'est-à-dire, fait \tentt\fam=\ttfam (p. 109)
\ttfam famille machine à écrire en mathématique (p. 218)

`\ttraggedright` utilise des caractère de machine à écrire et autorise le marge droite des paragraphes à varier d'une ligne à l'autre (p. 122)

`\u` accent bref en texte, comme dans ŭ (p. 106)

`*\uccode` le code de caractère de la forme capitale d'une lettre (p. 109)

`*\uchyph` si positif, laisse césurer des mots qui commence avec une lettre capitale (p. 133)

`\underbar` underline the specified text without avoiding any descenders, comme dans fog (p. 169)

`\underbrace` produit une accolade couvrant le bas d'une formule, comme dans $\underbrace{x+x}$ (p. 210)

`*\underline` souligne une formule mathématique sous les descendantes, comme dans $\underline{x+y}$ (p. 210)

`*\unhbox` décharge le contenu de la boîte d'un registre de boîte spécifié de la liste courante et vide le registre ; légal en modes horizontaux seulement (p. 171)

`*\unhcopy` comme `\unhbox`, mais ne vide pas le registre (p. 171)

`*\unkern` si le dernier élément de la liste courante est un crénage, l'enlève (p. 177)

`*\unpenalty` si le dernier élément de la liste courante est une pénalité, l'enlève (p. 177)

`*\unskip` si le dernier élément de la liste courante est un ressort, l'enlève (p. 177)

`*\unvbox` décharge le contenu de la boîte d'un registre de boîte spécifié de la liste courante et vide le registre ; légal en modes verticaux seulement (p. 171)

`*\unvcopy` comme `\unvbox`, mais ne vide pas le registre (p. 171)

`\uparrow` relation : $\uparrow$ (p. 200)

`\Uparrow` relation : $\Uparrow$ (p. 200)

`\upbracefill` fill enclosing hbox with an upwards facing brace : $\overbrace{\qquad}$ (p. 220)

`\updownarrow` relation : $\updownarrow$ (p. 200)

`\Updownarrow` relation : $\Updownarrow$ (p. 200)

`\uplus` opérateur plus cupped : $\uplus$ (p. 197)

`*\uppercase` converti les lettres minuscules d'un texte spécifié en capitale (p. 110)

`\upsilon` lettre grecque mathématique υ (p. 195)

`\Upsilon` lettre grecque mathématique Υ (p. 195)

`\v` accent tchèque en texte, comme dans ŏ (p. 106)

`*\vadjust` produit du matériel en mode vertical après la ligne courante (p. 126)

`*\valign` aligne du texte dans des rangées (p. 185)

`\varepsilon` variante de la lettre grecque mathématique ε (p. 195)

`\varphi` variante de la lettre grecque mathématique φ (p. 195)

`\varpi` variante de la lettre grecque mathématique ϖ (p. 195)

\varrho variante de la lettre grecque mathématique ϱ (p. 195)
\varsigma variant de la lettre grecque ς (p. 195)
\vartheta variante de la lettre grecque mathématique ϑ (p. 195)
*\vbadness badness threshold for reporting underfull or overfull vboxes, par défaut 1000 (p. 176)
*\vbox produit une vbox dont la ligne de base est celle de la boîte du haut incluse (p. 167)
*\vcenter centre le texte spécifié sur l'axe mathématique (p. 221)
\vdash symbole left turnstile : $\vdash$ (p. 198)
\vdots points verticaux en mathématique : $\vdots$ (p. 211)
\vec accent vecteur en mathématique, comme dans $\vec{x}$ (p. 207)
\vee opérateur "ou" logique : $\vee$ (p. 197)
\vert relation barre : $|$ (p. 196)
\Vert relation double barre : $\|$ (p. 196)
*\vfil produit un ressort vertical infiniment étirable (p. 163)
*\vfill produit un ressort vertical encore plus infiniment étirable que celui produit par \vfil (p. 163)
*\vfilneg produit un ressort vertical négatif infiniment étirable (p. 165)
\vfootnote produit un note de bas de page spécifiée avec une marque de référence spécifiée, mais ne produit la marque de référence dans le texte (p. 151)
*\vfuzz space threshold for reporting overfull vboxes, par défaut 0.1 pt (p. 176)
\vglue produit un ressort vertical spécifié qui ne disparait pas sur des coupures de page (p. 162)
*\voffset offset vertical relatif d'un pouce par rapport au coin haut du papier (p. 146)
\vphantom produit un formule invisible sans largeur mais de hauteur et profondeur naturelle (p. 175)
*\vrule produit un filet vertical ; légal seulement en modes horizontaux (p. 178)
*\vsize hauteur de page, par défaut 8.9 in (p. 146)
*\vskip produit un ressort vertical spécifié (p. 161)
*\vsplit coupe le contenu d'un registre de boîte spécifiée de la hauteur spécifié (p. 155)
*\vss produit un ressort vertical qui est infiniment étirable et infiniment rétrécissable (p. 164)
*\vtop produit une vbox dont la ligne de base est celle de la boîte du haut englobée (p. 167)
*\wd la largeur d'une boîte dans un registre de boîte spécifié (p. 173)
\wedge opérateur logique "et" : $\wedge$ (p. 197)
\widehat accent mathématique, comme dans $y + \widehat{z} + a$ (p. 207)
\widetilde accent mathématique $b + \widetilde{c} + d$ (p. 207)
*\widowpenalty pénalité pour une seule ligne débutant une page, par défaut 150 (p. 144)

`\wlog` `\write` la liste de token spécifiée dans le fichierlog (p. 270)
`\wp` symbole Weierstraß 'p' : $\wp$ (p. 196)
`\wr` opérateur wreath product : $\wr$ (p. 197)
`*\write` écrit une ligne vers un flot de sortie spécifié (p. 258)
`*\xdef` équivalent à `\global\edef`, c'est-à-dire, défini globalement une macro, développant immédiatement le texte de replacement (p. 239)
`\xi` lettre grecque mathématique ξ (p. 195)
`\Xi` lettre grecque mathématique Ξ (p. 195)
`*\xleaders` produit des leaders avec l'espace en trop à gauche redistribué egalement entre les boîtes leader (p. 179)
`*\xspaceskip` si différent de zéro et `\spacefactor` $\geq$ 2000, surcharge le ressort inter-mot normal (p. 113)
`*\year` l'année courante, comme un nombre (p. 233)
`\zeta` lettre grecque mathématique ζ (p. 195)

GNU Free Documentation License

1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not

Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified

Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from

their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

11. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME.
> Permission is granted to copy, distribute and/or modify this document
> under the terms of the GNU Free Documentation License, Version 1.2
> or any later version published by the Free Software Foundation;
> with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
> A copy of the license is included in the section entitled "GNU
> Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the
> Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

Dans les entrées de cet index, un numéro de page en italique indique une entrée principale ou une définition.

␣ *3*, 57
\␣ 110, 111, 323
! 13
\! *222*, 323
\" *106*, 324
28, 46, *73*, 324
  code de catégorie 57
  en texte ordinaire 15
\# 15, *104*, 324
$ 16, 28, 100, 324
  code de catégorie 56
  en texte ordinaire 15
\$ 15, *104*, 324
$$ 16, 79
% 13, 28, 324
  code de catégorie 57
  en sortie 302
  en texte ordinaire 15
  pour éliminer des espaces non désirés 278
\% 15, *104*, 324
& 28, 45, 46, 324
  code de catégorie 56
  en texte ordinaire 15
\& 15, *104*, 324
' *196*, 324
\' *106*, 324
) 64
* *198*, 324
\+ 45–46, *181*, 324
\, *222*, 324
- 14
  mène à \hyphenpenalty 131
\- 55, *132*, 324
  sur une coupure de ligne 59, 276
-- 14
--- 14
. 13
\. *106*, 324
/ 64
\/ *112*, 324
\; *222*, 324
= 199
\= *106*, 324
\> *222*, 324
? 13
[64
\ 28, 55, 96, 324
  code de catégorie 56
  en texte ordinaire 15
] 64
^ 28, *205*, 324
  code de catégorie 57
  en texte ordinaire 15
\^ *106*, 324
^^? 57
^^@ 57
^^A 57
^^I 54, 57
^^J 54
^^K 57
^^L 57, 302, 324
^^M 54, *111*, 301, 324
  code de catégorie 57

_ 28, *205*, 324
code de catégorie 57
en texte ordinaire 15
_ 15, *104*, 324
\‘ *106*, 324
{ 28, *235*, 324
avec \expandafter 241
code de catégorie 56
débuter un groupe 15, 71
en texte ordinaire 15
\{ 64, *199*, 324
| 196
\| *196*, 324
} 28, *235*, 324
code de catégorie 56
en texte ordinaire 15
terminer un groupe 15, 71
\} 64, *199*, 324
~ 13, 28, 57, *111*, 276, 324
en texte ordinaire 15
sur une coupure de ligne 59
\~ *106*, 324

\aa *103*, 324
\AA *103*, 324
\above *208*, 324
\abovecolumnspenalty *318*
\abovedisplayshortskip *225*, 279, 324
\abovedisplayskip *225*, 279, 325
abovewithdelims 63
\abovewithdelims *209*, 325
\accent *106*, 325
accent aigu 106, 207
accent barre 207
accent barre dessous 207
accent bref 106, 207
accent cédille 106
accent circonflexe 106, 207
accent circonflexe inversé 106
accent circonflexe large 207
accent double point 207
accent grave 106, 207
accent macron 106
accent point 106, 207
accent point en dessous 106
accent tchèque 207
accent tie-after 106
accent tilde 106, 207
accent tilde large 207
accent tréma 106
accent tréma Hongrois 106
accent vecteur 207
accents 28, 106, 207
aligner des 220–221
accolades 210, 220
espace après 278
orphelines 283–284
accolades horizontales 220
actions répétées 248–249
\active 54, *259*, 325
\acute *207*, 325
\adjdemerits *131*, 325
\advance *253*, 325
\advancepageno *148*, 325
\ae *103*, 325
\AE *103*, 325
affichages
actions pour chaque affichage 226
multi-ligne 216–217
paramètres d'espacement pour 224–225
affichages, formatage 305
\afterassignment *237*, 325
\aftergroup *237*, 317, 325
aides au diagnostique 261–269
\aleph *196*, 325
alignements *45–48*
caractère tabulation pour 56
commandes pour 181–192
espace entre rangées d' 141
séquence de contrôle outer dans des 288
utiliser \offinterlineskip dans des 141
alignements tabulés 181–184
\allowbreak *127*, *142*, 325
\alpha *195*, 325
alphabets européens 103
\amalg *197*, 325
$\mathcal{AMS}$-TEX viii, 18
\AMSTeX *304*
Anatomie de TEX 16, *48–49*
\angle *196*, 325
\approx *198*, 199, 325
\arccos *201*, 325
\arcsin *201*, 325
\arctan *201*, 325
\arg *201*, 325
⟨*argument*⟩ 4
arguments 4, 11, *49–50*
arguments délimités 50
arguments non délimités 50
arithmétique 253–254
\arrowvert *220*, 325
\Arrowvert *220*, 325
ASCII 53, 56, 89
ASCII *50–51*
assignements *51*, 87
de boîtes 170

de registres 250
\ast *197*, 325
\asymp *198*, 199, 325
\atop *208*, 325
\atopwithdelims 63, *209*, 325
autres caractères 54, 57
axe 221

\b *207*, 325
backslash 234
\backslash 64, *196*, 325
\badness 18, *176*, 325
\bar *207*, 325
barre de révision 32
basculement en mathématique 56
\baselineskip 95, *139–140*, 141, 326
et \smallskipamount, etc. 161
et préservation de fin de page 276
\batchmode *261*, 326
\begingroup *235*, 326
\beginsection *135*, 326
\belowdisplayshortskip *225*, 326
\belowdisplayskip *225*, 326
beta 196
\beta *195*, 326
\bf *109*, 326
\bffam *218*, 326
\bgroup *235*, 326
bibliographies 14
BIBTEX 19
\BibTeX *304*
\big *219*, 326
\Big *219*, 326
\bigbreak *143*, 326
\bigcap *202*, 326
\bigcirc *197*, 326
\bigcup *202*, 326
\bigg *219*, 326
\Bigg *219*, 326
\biggl *219*, 326
\Biggl *219*, 326
\biggm *219*, 326
\Biggm *219*, 326
\biggr *219*, 326
\Biggr *219*, 326
\bigl *219*, 326
\Bigl *219*, 326
\bigm 197, *219*, 326
\Bigm *219*, 326
\bigodot *202*, 326
\bigoplus *202*, 326
\bigotimes *202*, 326
\bigr 64, *219*, 326
\Bigr *219*, 326
\bigskip 143, *160*, 326
\bigskipamount *161*, 327
\bigsqcup *202*, 327
\bigtriangledown *197*, 327
\bigtriangleup *197*, 327
\biguplus *202*, 327
\bigvee *202*, 327
\bigwedge *202*, 327
\binoppenalty *132*, 327
\blackbox *304*
\bmod *201*, 327
boîtes 17, *51–53*
copie 170, 171
déplacement 172–173
dernière boîte dans une liste 177
dessiner des 32
extraction de contenu de 171
extraction du contenu de 171
fantôme 174–175
hauteur de 71
invisible 97
largeur de 72
lignes de base de 72
overfull 175–177, 276–278
point de référence de 88
profondeur de 90
ressort avec 93
tester si elle est vide 246
underfull 175–177, 276–278
vides 175
\bordermatrix *213*, 327
\bot *196*, 327
\botmark 77, 78, *150*, 327
bouche 16, *48*
Voir aussi Anatomie de TEX
boucles 248–249
\bowtie *198*, 327
\box *170*, 171, 327
\boxmaxdepth 53, 168, *169*, 327
\box255 85, 91, 150, 154
\brace *208*, 327
\bracevert *220*, 327
\brack *208*, 327
\break 59, *126*, *142*, 327
corriger des coupures de ligne avec 276
fin de ligne sur 286
\breve *207*, 327
\brokenpenalty *145*, 327
\buildrel *210*, 327
\bullet *197*, 327
\bye *255*, 327

\c *106*, 327
cadré à droite 95, 115, 121, 316–317
cadré à gauche 95, 115, 122, 316–317

\cal *217*, 327
\cap *197*, 327
caractère d'échappement *55*, 83, 96, 98
code de catégorie 56
représenté par \escapechar 234
caractères 50, *53–54*, 105
code de catégorie 56
codes ASCII pour 84
définis par \chardef 240
spéciaux 28
caractères actifs *54–55*, 57
caractères de contrôle 7, 50, 54
caractères entrées 288
caractères espace 24, 303
caractères espaces
code de catégorie 57
caractères ignorés 57
caractères imprimables 50
caractères invalides 57
caractères mathématiques 240
décrits par mathcodes 78
caractères spéciaux 104
caractéristiques de programmation 229
\cases *209*, 327
\catcode 54, 55, 56, *259*, 327
\cdot *197*, 204, 327
\cdotp *204*, 327
\cdots *211*, 327
\center *316*
\centereddisplays *305*
\centerline 95, *115*, 328
centimètre 62
centrage 34, 115
césure 34, *55*, 132–135, 145
allemande 133
pénalités pour 131–132
césures optionnelles 55, 59, 131, 132
mauvaises coupures de ligne, corriger avec 276
overfull boxes, corriger avec 277
champignons 38
\char 54, 56, *105*, 328
\chardef *240*, 242, 328
\check *207*, 328
\chi *195*, 328
chiffres hexadécimaux 54
chiffres romain 232
\choose *208*, 328
cicéro 62
\circ *197*, 328
classe *56*, 218, 226
d'un délimiteur 64
\cleaders 91–93, *179*, 328
\cleartabs *184*, 328
\closein 68, *256*, 328
\closeout 68, *257*, 328
avec \immediate 258
élément extraordinaire produit par 66
\clubpenalty *144*, 328
\clubsuit *196*, 328
cmex10 220
cmtt10 116
⟨*code de caractère*⟩ 4
codes de catégorie *56–58*
attaché pendant l'entrée 53
cause d'espaces non désirés 278
changement 288–289
de caractères actifs 54
définitions pratiques pour 301
en table de \catcode 259
pour texte verbatim 286
tester 243
codes délimiteurs 64–65, 215, 260
\colon *204*, 328
colonne 184
combinaisons, notation pour des 209
commandes 3, 10–11, *58*
arguments de 49
versus séquences de contrôle 11
commandes de boîtes 166–177
commentaires *13*, 24, 57
condition incomplète 288
\cong *198*, 199, 328
constantes décimales *59*, 84
constructeur de page 85
construction de page *59*
contrôler TEX 260–261
contributions récentes 85
conversion de casse 109–110
\coprod *202*, 328
\copy *170*, 328
\copyright *104*, 328
\cos *201*, 328
\cosh *201*, 328
\cot *201*, 328
\coth *201*, 328
couleurs de carte à jouer 28, 196
\count 49, *250*, 254, 328
\countdef *253*, 328
\count0 148
coupures de ligne 17, 48, *59–60*, 126–135, 280
crénages sur des 163
dans des formules mathématiques 127
démérites pour 65
effacer une 13

encouragement ou découragement 126–128
et formation de paragraphe 120–126
mauvaises 276
médiocrité pour 80
paramètres affectant les 128–132
tracer 267
coupures de page *60–61*, 142–146
crénages sur des 163
en liste dédoublées 156
encourager ou décourager 142–144
insérées par l'estomac de TEX 17
insertions sur des 72
mauvaises 273–276
médiocrité pour 80
paramètres de 144–146
ressort sur 276
tracer 267
\cr 45–48, *186*, 187, 328
\crcr *187*, 328
crénages *61*, 163
comme éléments de liste 51
créer de l'espace avec 67
dans des formules mathématiques 223
dernier crénage dans une liste 177
\csc *201*, 328
\csname 234, *241*, 328
développé par les règles d'\edef 239
\cup *197*, 328

\d *106*, 328
\dag *104*, 328
\dagger *197*, 328
\dashv *198*, 328
date 233, 307
\day *233*, 307, 328
\ddag *104*, 328
\ddagger *197*, 328
\ddot *207*, 329
\ddots *211*, 329
\deadcycles *155*, 329
debugger 261–269
\def 97, *238*, 329
rendre global 70
\defaulthyphenchar *135*, 329
\defaultskewchar *221*, 329
\deg *201*, 329
\delcode 64, 215, *260*, 329
\delimiter 64, *213*, 329
\delimiterfactor *213*, 329
\delimitershortfall *213*, 329
délimiteurs *63–65*, 199–200, 212–213
élargis 219
hauteur des 213
nuls, espace pour 225
parties de 220
\delta *195*, 329
\Delta *195*, 329
démérites 59, *65*
Desgraupes, Bernard 20
\det *201*, 205, 329
développer des tokens 16
\diamond *197*, 329
\diamondsuit *196*, 329
point didôt 62
\dim *201*, 329
\dimen *250*, 254, 329
\dimendef *253*, 329
⟨*dimension*⟩ 4, 284
dimensions *61–62*
comparer 245
de registres de boîte 173
maximum 252
négatives 62
dimensions de page 146
\discretionary *132*, 329
\displayindent *224*, 329
\displaylimits *203*, 329
\displaylines *216*, 306, 329
\displaysetup 306
\displaystyle 98, *206*, 329
\displaywidowpenalty *144*, 329
\displaywidth *224*, 329
dispositifs de sortie 8
disposition de page *62–63*
\div *197*, 329
\divide *254*, 329
\dot *207*, 329
\doteq *198*, 329
\dotfill *180*, 329
\dots *105*, 329
double colonnes 319
double espacement 140
\doublehyphendemerits *131*, 329
\downarrow 64, *200*, 330
\Downarrow 64, *200*, 330
\downbracefill *220*, 330
\dp *173*, 330
drivers. *Voir* drivers de périphérique
drivers d'impression 258
drivers de périphérique 8, 89
instructions par \special 258
origine de la page connu des 146
droite déchirée 99
\dump 69, *271*, 330
fichier .dvi 8
boîte enregistré dans le fichier 266

comme un fichier résultat 68
convertit par driver 89
créé par l'intestin de TEX 16, 49
élément extraordinaire en 66
matériel de la routine de sortie 60, 85, 96
matériel inséré par `\special` 66
réception de boîtes de `\shipout` 154

écrire un fichier 258
`\edef` *238*, 330
expansion de `\'`*c* en 84
rendre global 70
éditeur de texte 7
`\egroup` *235*, 330
`\ehrule` *302*
`\eject` *143*, 274, 330
El Paso 99
élément extraordinaire *66*, 134
éléments *65*
`\ell` *196*, 330
`\else` 99, *248*, 330
`em` 62
`\emergencystretch` 18, *129*, 277, 330
`\empty` *250*, 330
`\emptyset` *196*, 330
en 160
`\end` *255*, 330
`\endcsname` 330
`\endgraf` *117*, 330
`\endgroup` *235*, 330
`\endinput` *255*, 330
`\endinsert` *153*, 330
`\endline` *187*, 330
`\endlinechar` *260*, 289, 330
`\enskip` *160*, 330
`\enspace` *160*, 330
entêtes 62–63, *66*
marques utilisés en 77
multi-lignes 282–283
entêtes de section 135
entrée (colonne ou rangée) 45, 184–185
énumérations 308–310
`\environment` *315*
environnements 314
`eplain.tex` 19, 301–321
`\epsilon` *195*, 330
`\eq` *305*
`\eqalign` *216*, 330
`\eqalignno` *216*, 306, 330
`\eqdef` *314*
`\eqn` *305*
`\eqno` *215*, 330
`\eqprint` *314*
`\eqref` *314*
équations, labelisation 314
`\equiv` *198*, 199, 330
`\errhelp` *269*, 270, 330
`\errmessage` *269*, 330
développé par les règles d'`\edef` 239
`\errorcontextlines` 18, *270*, 298, 330
`\errorstopmode` *260*, 331
`\escapechar` 55, *234*, 258, 331
espace 12–13, *66–67*
dans des formules mathématiques 222–224
inter-mots 93, 110–114
non désiré 278–279
perdu 278
production d' 159–165
visible *3*
espace blanc, préserver de l' 286
espace contrôlé 10, 110, 278
espace horizontal 159–160, 161–165
espace non désiré 12
espace supplémentaire 222, 266
espace vertical 160–165
réserver en haut d'une page 276
espace visible 110
espacement 113
ajuster avec des crénages 61
inter-ligne 30
espacement de ligne 139
espacement inter-mots 79, 112, 113
estomac 16, *48*
Voir aussi Anatomie de TEX
`\eta` *195*, 331
étirement *67*, 93–95
`\everycr` 186, *192*, 331
`\everydisplay` *226*, 306, 308, 331
`\everyfootnote` *319*
`\everyhbox` *170*, 331
`\everyjob` *271*, 331
`\everymath` *226*, 331
`\everypar` 86, 116, *119*, 331
pour fixer `\looseness` 130
pour indentation rémanente 124
`\everyvbox` *170*, 331
`\evrule` *302*
`ex` 62
exécuter TEX 9, 260–261
`\exhyphenpenalty` *131*, 331
`\exists` *196*, 331
`\exp` *201*, 331
`\expandafter` 238, *241*, 331
exposants 57, 205–206
extensions mathématiques 67

facteur 62
facteur d'échelle 231
\fam *218*, 331
famille *67*
comme partie de mathcode 78
donnée par \fam 218
réservée par \newfam 252
taille script en 98
taille scriptscript en 99
taille texte en 99
variable 79
fantômes 174–175
fermant 199
\fi 99, *248*, 331
fichier de forme 76, 89
fichier de formes 8
fichier de métriques
caractère oblique par défaut dans le 221
fichier de pixel 89
fichier format *69*, 271
fichier log *69*
écrire par \wlog 269
écrit par \write 258
message d'erreur 293
tracer des statistiques dans le 268
fichier résultat 68
fichier source 9, 48, 68
fichiers *67–69*, 255–259
tester l'existence de 305
fichiers auxiliaires 234
fichiers de métriques 8, 65, 229
correction italique dans 112
tiret de césure par défaut dans 135
fichiers de polices 8
fichiers de sortie 257–259
fichiers d'entrée 255–257
imbriqué 9
fil 62, 95
\filbreak *143*, 274, 331
filets *69–70*, 178–179, 281–282
épaisseur de 302
filets horizontaux 69–70, 178–179
filets verticaux 69–70, 178–179
fill 62, 95
filll 62, 95
fin de fichier, tester 247
fin de ligne 57, 66, 111
fin d'exécution 255
\finalhyphendemerits *132*, 331
\firstmark 77, 78, *150*, 331
\fivebf *108*, 331
\fivei *108*, 331
\fiverm *108*, 331
\fivesy *108*, 331
flèches 181, 200–201, 210
\flat *196*, 331
\floatingpenalty *145*, 331
flots de sortie 68, *70*
écriture 258
fermeture 257
ouverture 257
réservés par \newwrite 252
flots d'entrée *70*
lecture avec \read 256
lire avec \read 68
ouverture 256
réservés par \newread 252
\flushleft *316*
\flushright *316*
\fmtname *233*, 331
\fmtversion *233*, 331
\folio *149*, 331
fonctions, noms de 201–202
\font 135, 221, *229*, 303, 332
\fontdimen *230*, 303, 332
\fontname *235*, 332
\footline 63, 88, *149*, 282, 332
\footnote 72, *151*, 153, 332
\footnotemarkseparation *318*
\for 303
for loop 303
\forall *196*, 332
\frac *304*
fractions 208–211
en forme de slash 304
produites par \over 208
\frenchspacing 14, *112*, 332
\frown *198*, 332
\futurelet *240*, 332

\gamma *195*, 332
\Gamma *195*, 332
gauche déchirée 99
\gcd *201*, 332
\gdef 71, 236, *239*, 332
\ge *198*, 199, 332
\generaldisplay 306
\geq *198*, 199, 332
\gets *200*, 332
fichier .gf 8, 89
\gg *198*, 332
global *70–71*
\global 70, *236*, 332
\globaldefs 70, *236*, 289, 332
\gobble *301*
\gobblethree *301*
\gobbletwo *301*
\goodbreak *143*, 332

Goossens, Michel 20
\grave *207*, 332
groupes 15–16, 56, *71*, 235–237

\H *106*, 332
\halign 46–47, *184*, 186, 332
grouper pour 16
illégal dans le mode mathématique 306
vertical par nature 81
\hang *123*, 280, 332
\hangafter *123*, 125, 332
\hangindent 86, *123*, 125, 280, 332
\hat *207*, 332
hauteur 52, *71*, 173
\hbadness 129, *176*, 332
\hbar *196*, 332
hbox 51, *71*
construction avec \hbox 166, 167
contrôler des coupures de lignes 59
mode horizontal pour 81
tester 246
\hbox 51, *166*, 276, 332
overfull box provenant de 278
\headline 63, 66, *149*, 282, 332
\heartsuit *196*, 332
heure du jour 233, 307
\hfil *163*, 165, 166, 278, 332
\hfill *163*, 333
\hfilneg *165*, 333
\hfuzz 129, *176*, 277, 333
\hglue *162*, 163, 333
\hidewidth *191*, 333
\hoffset 63, 77, *146*, 333
\holdinginserts 18, *155*, 333
\hom *201*, 333
\hookleftarrow *200*, 333
\hookrightarrow *200*, 333
\hphantom *175*, 333
\hrule 69–70, *178*, 281, 333
vertical par nature 81
\hrulefill *180*, 333
\hsize 63, 77, *120*, *146*, 333
déterminé par \magnification 231
\hskip 93, *161*, 333
\hss *164*, 278, 333
\ht *173*, 333
\hyphenation 55, *133*, 333
\hyphenchar *134*, 135, 242, 333
\hyphenpenalty *131*, 145, 333
hypothèses 136

\i *106*, 333
\ialign *186*, 333
\if *243*, 333
\ifcase *247*, 333
\ifcat *243*, 333
\ifdim *245*, 333
\ifempty *303*
\ifeof *247*, 333
\ifeqno *305*
\iff *200*, 333
\iffalse *247*, 333
\ifhbox *246*, 333
\ifhmode *246*, 334
\ifinner *246*, 334
\ifleqno *305*
\ifmmode *246*, 334
\ifnum *245*, 334
\ifodd *245*, 334
\iftrue *247*, 334
\ifvbox *246*, 334
\ifvmode *246*, 334
\ifvoid 153, *246*, 334
\ifx *244*, 334
\ignorespaces *260*, 334
\Im *196*, 334
\imath 106, *196*, 334
\immediate 66, 69, 257, *258*, 334
imprimantes 8
\in *198*, 334
\indent 86, *117*, 334
indentation 26, 117–125
indentation rémanente 123
indices 57, 205–206
\inf *201*, 334
information locale *7*, 9
\infty *196*, 334
initex 69, 271
\input 7, 48, 68, *255*, 334
\inputlineno *256*, 334
\insert 72, *153*, 334
<inserted text> 296
insertions *72*
commandes pour 152–154
forcés par \supereject 143
numéros réservés par \newinsert 252
pénalités pour 145
\insertpenalties *145*, 334
\int *202*, 203, 204, 334
limites après 203
\interlinepenalty *144*, 334
intestin 16, 48, *49*
Voir aussi Anatomie de TEX
\iota *195*, 334
\it *109*, 334
italique 112

\item *136*, 334
\itemitem *136*, 334
\itfam *218*, 334

\j *106*, 334
Jean-Côme Charpentier 20
\jmath 106, *196*, 334
\jobname *233*, 334
\jot *223*, 334
justification *99–100*, 115, 122
justification à droite 34
justification à gauche 34

\kappa *195*, 334
\ker *201*, 334
\kern *163*, 335
Kluth, Marie-Paule 20
Knuth, Donald E. 18, 20, 304

\l *103*, 335
\L *103*, 335
\lambda *195*, 335
\Lambda *195*, 335
Lamport, Leslie 18
\land *197*, 335
\langle 64, *199*, 335
\language 18, 66, *134*, 335
langues étrangères 18
Voir aussi langues européennes
langues européennes 72, 134
largeur 52, *72*, 173
\lastbox *177*, 178, 335
\lastkern *177*, 242, 335
\lastpenalty *177*, 242, 335
\lastskip *177*, 242, 335
LaTeX viii, 18, 233
\LaTeX *304*
\lbrace 64, *199*, 335
\lbrack 64, *104*, *199*, 335
\lccode *109*, 335
\lceil 64, *199*, 335
\ldotp *204*, 335
\ldots *211*, 335
\le *198*, 199, 335
\leaders 91–93, *179*, 335
\left 63, *212*, 335
\leftarrow *200*, 335
\Leftarrow *200*, 335
\leftarrowfill *181*, 335
\leftdisplays *305*, *306*
\leftharpoondown *200*, 335
\leftharpoonup *200*, 335
\lefthyphenmin 18, *134*, 335
\leftline 95, *115*, 335
\leftrightarrow *200*, 335
\Leftrightarrow *200*, 335
\leftskip 99, *121*, 280, 335
lemmes 34, 136
\leq *198*, 199, 335
\leqalignno *216*, 306, 336
\leqno *215*, 336
\let 97, *240*, 336
\letreturn *301*, 303
\letter *301*
lettre 54, 57
lettres grecques 195–196, 218
lettres sans points 106
\lfloor 64, *199*, 336
\lg *201*, 336
\lgroup *212*, 336
\li *308*
ligatures *72*, 103–104, 107
ligne 120
ligne d'entrée 260
ligne orpheline 144
ligne veuve 144
lignes de base 30, 52, *72–73*, 88
\lim *201*, 205, 336
\liminf *201*, 336
limites 202
limites inférieures 205
limites supérieures 205
\limits *203*, 336
\limsup *201*, 336
\line *115*, 336
\linepenalty *130*, 336
\lineskip 95, *139–140*, 336
\lineskiplimit 95, *139–140*, 336
lire un fichier 256
\listcompact *308*
liste verticale principale 85
listes *73*
listes d'éléments 34, 136, 308–310
listes de description 32
listes horizontales 51, *73*, 93
filet en 70
hbox formés à partir de 71
ne peuvent contenir de commandes verticales 81
pénalités dans 87
listes verticales 51, *73*, 93
filet en 70
insérer dans des paragraphes 126
ne peuvent contenir de commandes horizontales 82
pénalités dans 87
vbox formées à partir de 100
\listing *310*
livre sterling 104
\ll *198*, 336

\llap *115*, 336
\lmoustache *220*, 336
\ln *201*, 336
\lnot *196*, 336
\log *201*, 336
fichier .log 323
comme un fichier résultat 68
\loggingall *302*
logos 88
\long *239*, 244, 336
\longleftarrow *200*, 336
\Longleftarrow *200*, 336
\longleftrightarrow *200*, 336
\Longleftrightarrow *200*, 336
\longmapsto *200*, 336
\longrightarrow *200*, 336
\Longrightarrow *200*, 336
\loop *248*, 336
\looseness 86, *130*, 275, 336
\lor *197*, 337
\lower 52, *172*, 337
\lowercase *110*, 337
\lq *104*, 337

macros *73–76*, 238–250
arguments de 49, 289
contrôler leur développement 241–243
dans des fichiers auxiliaires 7
définir des 238–240
développées dans l'estomac de TEX 48
globale 70
nommées par des caractères actifs 54
outer 84–85, 288
paramètres de 49, 57, 73–75, 289
rendre lisible des 289–290
tracer des 266
utiliser \begingroup et \endgroup dans des 235
utiliser \bgroup et \egroup dans des 236
\mag 62, 76, *231*, 337
magnification 8, 62, *76*
\magnification *231*, 337
\magstep 76, 231, *232*, 337
\magstephalf 76, 231, *232*, 337
majuscules
conversion en 109–110
\makeactive *301*
\makeblankbox *304*
\makecolumns *317*
\makefootline 283
\makeheadline 282
\mapsto *200*, 337
marge de droite 93
marge du bas 93
marges 26, 62–63, *77*
\mark 77, 78, *150*, 337
marque de référence 151
marques *77–78*, 150–151
avec entêtes ou pieds de page 63
pour listes dédoublées 150
marques de citation 14, 24
marques diacritiques. *Voir* accents
matériel flottant 145, 152
\mathaccent *207*, 337
\mathbin *226*, 337
\mathchar *105*, 337
\mathchardef *240*, 242, 337
\mathchoice *206*, 337
\mathclose *226*, 337
\mathcode 79, 213, *259*, 260, 337
mathcodes *78–79*, 240
classe encodée en 56
mathématiques 16, 40, 42, 195–227
accents 207
mathématiques affichées 16, *79*, 144, 279
\mathinner *226*, 337
\mathop *226*, 337
\mathopen *226*, 337
\mathord *226*, 337
\mathpalette *206*, 337
\mathpunct *226*, 337
\mathrel *226*, 337
\mathstrut 97, *174*, 337
\mathsurround *225*, 337
matrice 213
\matrix *213*, 337
\max *201*, 337
\maxdeadcycles *155*, 337
\maxdepth *147*, 156, 338
\maxdimen *252*, 338
\meaning *234*, 338
\medbreak *143*, 338
médiocrité 65, *79–80*
\medmuskip *222*, 338
\medskip 143, *160*, 338
\medskipamount *161*, 338
\message *269*, 338
développé par les règles d'\edef 239
message d'aide 270
messages d'erreur 9, 269–270, 293–298
messages, envoyer 269–270
METAFONT vii, 36
\MF *304*

\mid *198*, 338
\midinsert 72, *152*, 153, 338
millimètre 62
\min *201*, 338
minuscules
conversion en 109–110
\mit *217*, 338
\mkern *223*, 338
mode horizontal *81*
filets en 178
tester 246
mode horizontal ordinaire 80, 81
mode horizontal restreint 80, 81
mode mathématique *81–82*
tester 246
mode mathématique d'affichage 81
mode mathématique de texte 80, 81
mode mathématique hors-texte 81
mode ordinaire *82*
mode restreint *82*
horizontal 281
tester 246
mode vertical 81, *82*
filets en 178
tester 246
mode vertical ordinaire 80, 82
mode vertical restreint 80, 82
\models *198*, 338
modes 17, *80–81*
modulo 201, 202
\month *233*, 307, 338
\monthname *307*
mots de contrôle 10, *83*, 96
mots trop rapprochés 278
\moveleft 52, 167, *172*, 338
\moveright 52, 167, *172*, 338
\mp *197*, 338
\mskip 83, *223*, 338
mu 83, 251
\mu *195*, 338
muglue *83*
\multiply *254*, 338
\multispan *188*, 338
\muskip *250*, 254, 338
\muskipdef *253*, 338

\nabla *196*, 338
\narrower *121*, 280, 338
\natural *196*, 338
\ne *198*, 338
\nearrow *200*, 338
\neg *196*, 339
\negthinspace *159*, 339
\neq *198*, 339
\newbox 53, 91, *252*, 339
\newcount 91, 250, *252*, 339
\newdimen 91, *252*, 339
\newfam 67, *252*, 339
\newhelp *270*, 315, 339
\newif *248*, 339
\newinsert 153, *252*, 339
\newlanguage 18, *252*, 339
\newlinechar *259*, 269, 270, 339
\newmuskip 91, *252*, 339
\newread 68, *252*, 339
\newskip 91, *252*, 339
nouveau TEX 18, 162
\newtoks 91, *252*, 339
\newwrite 68, *252*, 339
\ni *198*, 339
\noalign 47, 187, *189*, 339
\noboundary 18, 72, *107*, 339
\nobreak 59, *127*, *142*, 274, 339
\noexpand 238, *242*, 339
\noindent 86, *118*, 339
\nointerlineskip *141*, 339
\nolimits *203*, 339
⟨*nombre*⟩ 4
nombres *83–84*
comparer 245
convertir en caractères 232–233
tester la parité 245
nombres hexadécimaux 83
nombres octaux 83
noms de fichier *83*
\nonfrenchspacing *112*, 339
\nonscript *223*, 226, 339
\nonstopmode *261*, 339
\nopagenumbers *148*, 339
\normalbaselines *140*, 339
\normalbaselineskip *140*, 339
\normalbottom *144*, 340
\normallineskip *140*, 340
\normallineskiplimit *140*, 340
\not 340
notes de pied de page 24, 318
utiliser \textindent avec 118
\notin *198*, 340
\nu *195*, 340
\null *175*, 340
\nulldelimiterspace 65, 212, *225*, 340
\nullfont *108*, 340
\number 84, *232*, 340
\numberedfootnote *318*
\numberedlist *308*
\numberedmarker *309*
\numbername *305*
numéro de version 233
numéros d'équation 216

numérotation de page 91, 148–149
\nwarrow *200*, 340

\o *103*, 340
\O *103*, 340
\obeylines *128*, 286, 303, 340
\obeyspaces *113*, 128, 286, 303, 340
\obeywhitespace 113, 286, *303*
\odot *197*, 340
\oe *103*, 340
\OE *103*, 340
œsophage *48*, 56
\offinterlineskip 97, *141*, 173, 340
\oint *202*, 340
\oldstyle *217*, 340
\omega *195*, 340
\Omega *195*, 340
omicron 196
\ominus *197*, 340
\omit *187*, 340
\openin 68, *256*, 340
\openout 68, *257*, 340
avec \immediate 258
élément extraordinaire produit par 66
\openup *141*, 189, 340
opérateurs 132
grands 202–204
opérations 197–198
\oplus *197*, 340
\or 340
origine. *Voir* origine de la page
origine de la page 146
\oslash *197*, 340
\other *301*
\otimes *197*, 340
outer *84–85*, 239
\outer 84, *239*, 244, 288, 340
\output 96, *154*, 341
<output> 298
\outputpenalty *155*, 341
ouvrant 199
\over *208*, 212, 341
\overbrace *210*, 220, 341
overfull boxes 129, 276–278
\overfullrule *175*, 341
\overleftarrow *210*, 341
\overline *210*, 341
\overrightarrow *210*, 341
\overwithdelims 63, *209*, 341
\owns *198*, 341

\P *104*, 341
\pagedepth *145*, 341
\pagefilllstretch *146*, 341
\pagefillstretch *146*, 341
\pagefilstretch *146*, 341
\pagegoal *145*, 146, 156, 341
\pageinsert 72, *152*, 153, 341
\pageno *148*, 149, 341
pages 17, *85–86*
assemblées dans l'estomac de TEX 48
\pageshrink *146*, 341
\pagestretch *146*, 341
\pagetotal *145*, 341
\par *116*, 117, 341
d'une ligne vide 67, 111
dans des arguments de macro 239
en changeant la forme du paragraphe 281
terminer un paragraphe avec 86
paragraphes *86–87*
débuter des 116
étroit 26, 121
formation 116–126
forme 280–281
indenter. *Voir* indentation
ressort à la fin des 117
ressort entre 147
terminer un 12, 24
\parallel *198*, 341
paramètres *87*
assignements à des 51
comme commandes 4, 12
comme registres 90
délimités 75
et arguments 49
indiqués par # 57
non délimités 74
utiliser \the avec 242
parenthèses 63, 212
\parfillskip *117*, 341
\parindent 4, 87, *119*, 341
indentation pour listes d'éléments 136
\parshape 86, *124*, 342
\parskip 117, 118, 119, *147*, 275, 278, 342
\partial *196*, 342
Patashnik, Oren 19
patron *46*, 184–189
\pausing *261*, 342
pénalités
dernière pénalité dans une liste 177
en listes horizontales 59
en listes verticales 60
penalties *87–88*
\penalty *127*, *142*, 342

\percentchar 302
\perp *198*, 342
\phantom *174*, 342
\phi *195*, 342
\Phi *195*, 342
\pi *195*, 342
\Pi *195*, 202, 342
pica 62
pieds de page 62–63, *88*
marques utilisés en 77
multi-lignes 282–283
pilotes de périphériques 230
fichier .pk 8, 89
\plainoutput *154*, 342
plain TEX 3, 8, 9, *88*
familles de police en 67
\pm *197*, 342
\pmatrix *213*, 342
\pmod *202*, 342
poésie, composer de la 128
point 12, 13, 62
point d'échelle 62
point d'interrogation 13
point de référence 52–53, *88*
point décimal 84
point d'exclamation 13
points 105, 211
points de suspension 105
police Computer Modern 36, 51, 88, 116
police de type machine à écrire 116
police Palatino 36, 88
polices 28, *88–90*, 108, 217
caractères de césure pour 134–135
composites 284–285
familles de 67
nommer et modifier 229–232
noms de 235
paramètres de 230
polices résidentes 230
ponctuation 13, 24
ponctuation et formules de math. 204
\postdisplaypenalty *144*, 342
pouce 62
\Pr *201*, 342
préambule *46*
préambule 184, 185
\prec *198*, 199, 342
\preceq *198*, 199, 342
préchargée 69
\predisplaypenalty *144*, 342
\predisplaysize *224*, 342
\pretolerance *128*, 342
\prevdepth 139, *140*, 342
\prevgraf *126*, 342
préviewer 8
\prime *196*, 342
primitive *90*
commande 3, 49, 323
séquence de contrôle 97
\proclaim *136*, 342
\prod *202*, 342
profondeur 52, *90*, 173
programmes informatiques, composer des 128, 285, 288
\propto *198*, 342
\psi *195*, 342
\Psi *195*, 342
punaise 59
fichier .pxl 8

\qquad *160*, 342
\quad *160*, 342

règles de césure 134
règlures 179–181
\radical *215*, 343
\raggedbottom *144*, 343
\raggedright 99, *122*, 277, 343
\raise 52, *172*, 343
rangée 185
\rangle 64, *199*, 343
\rbrace 64, *199*, 343
\rbrack 64, *104*, *199*, 343
\rceil 64, *199*, 343
\Re *196*, 343
\read 68, *256*, 343
\readreffile *313*
\readtocfile *311*
références croisées 312
registre dimension 251
⟨*registre*⟩ 4
registres *90–91*, 250–254
arithmétique dans des 253–254
assignement à des 51
avec \the 242
paramètres comme 5, 12, 87
réserver 252–253
registres count 250
réservés par \newcount 252
registres de boîte 53, 90, 170–172, 173
réservés par \newbox 252
registres de compteur 90
registres de dimension 90
réservés par \newdimen 252
registres de muglue 90
registres de ressort 90
registres de token 90
registres muskip 251

réservés par \newmuskip 252
registres skip 251
réservés par \newskip 252
registres token 251
réservés par \newtoks 252
réglures *91–93*
relations 132, 198–199
mettre des formules sur des 210
\relax 9, *249*, 343
\relpenalty *132*, 343
remplissage 180, 181
\repeat *248*, 343
ressort 17, *93–95*, 223
créer de l'espace avec 67
dernier élément de ressort dans une liste 177
étirable infiniment 163–164
infiniment étirable 95
mathématique 83
négatif 165
ressort horizontal 161, 162
ressort inter-ligne *95*, 139–140
ressort interligne 52
ressort vertical 161, 162
⟨*ressort*⟩ 4
retour chariot 57
rétrécissement 93–95
rétrécissements *95*
révision vii
\rfloor 64, *199*, 343
\rgroup *212*, 343
\rho *195*, 343
\right 63, *212*, 343
\rightarrow *200*, 343
\Rightarrow *200*, 343
\rightarrowfill *181*, 343
\rightharpoondown *200*, 343
\rightharpoonup *200*, 343
\righthyphenmin 18, *134*, 343
\rightleftharpoons *200*, 343
\rightline 95, *115*, 343
\rightskip 99, *121*, 343
\rlap *115*, 343
\rm *109*, 343
\rmoustache *220*, 343
Rolland, Christian 20
\romannumeral 84, *232*, 343
\root *215*, 343
routine de sortie *96*, 153, 154–155
insertions, traitement de 72
par défaut dans plain TEX 154
signification de \insertpenalties dans la 145
\rq *104*, 343

\S *104*, 344
saut de paragraphe 147
saut horizontal 161
saut vertical 161
\sb *205*, 344
\scriptfont 67, *219*, 344
\scriptscriptfont 67, *219*, 344
\scriptscriptstyle 98, 205, *206*, 344
\scriptspace *226*, 344
\scriptstyle 98, 205, *206*, 344
\scrollmode *261*, 344
\searrow *200*, 344
\sec *201*, 344
séquence de contrôle interdite 288
séquences de contrôle 10–11, *96–97*
absorbent les espaces 10
comme tokens 16
convertir en chaines 234
définies avec \let 240
versus commandes 11
Seroul, Raymond 20
\setbox *170*, 344
\setlanguage 18, 66, *134*, 344
\setminus *197*, 344
\settabs 45–46, *182*, 344
\setuplistinghook *310*
\sevenbf *108*, 344
\seveni *108*, 344
\sevenrm *108*, 344
\sevensy *108*, 344
\sfcode *113*, 344
\sharp *196*, 344
\shipout 69, 96, *154*, 155, 268, 344
registres \count affichés avec 148
\show *261*, 288, 344
\showbox *261*, 264, 269, 344
\showboxbreadth 262, 266, *269*, 344
\showboxdepth 262, 266, *269*, 344
\showhyphens *133*, 344
\showlists *261*, 264, 269, 344
\showthe 91, *261*, 344
\sigma *195*, 344
\Sigma *195*, 202, 344
\sim *198*, 199, 344
\simeq *198*, 199, 344
\sin *201*, 345
\singlecolumn *319*
\sinh *201*, 345
\skew *220*, 345
\skewchar *221*, 242, 345
\skip *250*, 254, 345
\skipdef *253*, 345
\sl *109*, 345
slash 59, 128
\slash 59, *128*, 345
\slfam *218*, 345

\smallbreak *143*, 345
\smallint *202*, 345
\smallskip 143, *160*, 345
\smallskipamount *161*, 345
\smash *175*, 345
\smile *198*, 345
source, préparer 10
sous-formules empilées 208–211
\sp *205*, 345
\space *111*, 289, 345
\spacefactor *113*, 345
\spaceskip *113*, 345
\spacesub *303*
\spadesuit *196*, 345
\span *188*, 345
\special 66, *258*, 345
Spivak, Michael D. 19
\splitbotmark 78, *150*, 345
\splitfirstmark 78, *150*, 345
\splitmaxdepth *156*, 345
\splittopskip *156*, 345
\sqcap *197*, 345
\sqcup *197*, 345
\sqrt *214*, 345
\sqsubseteq *198*, 199, 345
\sqsupseteq *198*, 199, 346
\ss *103*, 346
\star *197*, 346
\string *234*, 241, 346
\strut 48, 97, *173*, 174, 346
struts *97–98*, 141, 173–174
dans des alignements verticaux 186
style d'affichage 98, 202, 206
style script 98, 205, 206, 219
style scriptscript 98, 205, 206, 219
style texte 98, 202, 206, 219
styles *98*, 206–207
styles de caractère 109, 218
\subset *198*, 199, 346
\subseteq *198*, 199, 346
\succ *198*, 199, 346
\succeq *198*, 199, 346
\sum *202*, 346
\sup *201*, 346
\supereject *143*, 346
support local 134
\supset *198*, 199, 346
\supseteq *198*, 199, 346
\surd *196*, 346
surimpression 116
\swarrow *200*, 346
symboles de contrôle 10, 96, *98*
symboles mathématiques 88, 206
symboles musicaux 28, 196
symboles ordinaires 196
symboles spéciaux 67, 103
système d'exploitation 67, 83

\t *106*, 346
\tabalign *181*, 346
table des matières 311
tables 45–48
Voir aussi alignements
\tabskip 47, *190*, 346
tabulation horizontale 57
tabulations 56, 182
taille d'origine 76
taille script 67, *98–99*
taille scriptscript 67, *99*
taille texte 67, *99*
\tan *201*, 346
\tanh *201*, 346
\tau *195*, 346
\tenbf *108*, 346
\tenex *108*, 346
\teni *108*, 346
\tenit *108*, 346
\tenrm *108*, 346
\tensl *108*, 346
\tensy *108*, 346
\tentt *108*, 346
terminal 262
test de cas 247
\testfileexistence *305*
tests conditionnels *99*, 243–248
\TeX *104*, 346
The TEXbook viii, 2, 45
TEX MEX *99*
texte centré 95, 99, 316–317
texte d'aide 270
texte de marque 77, 150
texte mathématique 16, *100*
texte recouvrant 115
texte verbatim 285
\textfont 67, *219*, 346
\textindent *118*, 346
\textstyle 98, *206*, 346
fichier .tfm 8, 61, 89, 229
\the 234, *242*, 250, 251, 347
théorèmes 34, 136
\theta *195*, 347
\Theta *195*, 347
\thickmuskip *222*, 347
\thinmuskip *222*, 347
\thinspace *159*, 347
tilde 13, 111
\tilde *207*, 347
\time *233*, 307, 347
\times *197*, 347

\timestamp *307*
\timestring *307*
tirets 14, 24
\to *200*, 347
<to be read again> 295
tokens 16, *100*
affichés par \show 261
assemblée à partir de caractères 48
comme commandes 58
montrer la signification de 234
passée à l'estomac de TEX 48
tokens de caractère 302
\toks *250*, 347
\toksdef *253*, 347
\tokstostring *302*
\tolerance *128*, 129, 130, 347
\top *196*, 347
\topglue 18, *162*, 276, 347
\topinsert 72, *152*, 153, 347
\topmark 77, *150*, 347
\topskip *147*, 156, 347
tracer 261–269
tracer des lignes. *Voir* filets
\tracingall *269*, 347
\tracingboxes *302*
\tracingcommands *264*, 347
\tracinglostchars *265*, 347
\tracingmacros *266*, 347
\tracingonline 262, *264*, 265, 266, 267, 268, 269, 347
\tracingoutput *266*, 269, 347
\tracingpages *267*, 347
\tracingparagraphs *267*, 347
\tracingrestores *268*, 347
\tracingstats *268*, 347
\triangle *196*, 347
\triangleleft *197*, 347
\triangleright *197*, 347
true 76
\tt *109*, 347
\ttfam *218*, 347
\ttraggedright *122*, 348
TEX Users Group 19
TUGBoat 19

\u *106*, 348
\uccode *109*, 348
\uchyph *133*, 348
\unbox 178
\uncatcodespecials *301*
\underbar *169*, 348
\underbrace *210*, 220, 348
underfull boxes 276–278
\underline *210*, 348
\unhbox *171*, 348
\unhcopy *171*, 348
unités de mesure 62, *100*
unités mathématiques 83, *100*
\unkern *177*, 348
\unorderedlist *308*
\unpenalty *177*, 348
\unskip *177*, 348
\unvbox *171*, 348
\unvcopy *171*, 348
\uparrow 64, *200*, 348
\Uparrow 64, *200*, 348
\upbracefill *220*, 348
\updownarrow 64, *200*, 348
\Updownarrow 64, *200*, 348
\uplus *197*, 348
\uppercase *110*, 348
\upsilon *195*, 348
\Upsilon *195*, 348

\v *106*, 348
\vadjust *126*, 274, 275, 348
\valign 47, *185*, 348
grouper pour 16
horizontal par nature 82
utilisé dans \makecolumns 318
\varepsilon *195*, 348
variantes étroites 98
\varphi *195*, 348
\varpi *195*, 348
\varrho *195*, 349
\varsigma *195*, 349
\vartheta *195*, 349
\vbadness *176*, 349
vbox 51, *100*
largeur déterminée par \hsize 120
mode vertical pour 82
ressort inter-ligne pour 139
tester 246
\vbox 51, 52–53, 100, *167*, 349
empêcher des coupures de page avec 273
overfull box provenant de 278
\vcenter *221*, 306, 349
\vdash *198*, 349
\vdots *211*, 349
\vec *207*, 349
\vee *197*, 349
vers, composer des 128
\vert 64, *196*, 349
\Vert 64, *196*, 349
\vfil *163*, 278, 349
nécessaire avec \eject 143
remplir une vbox 168
\vfill *163*, 274, 349

\vfilneg *165*, 349
\vfootnote *151*, 153, 318, 349
\vfuzz *176*, 278, 349
\vglue *162*, 163, 349
virtex 69, 271
\voffset 63, 77, *146*, 282, 349
\vphantom *175*, 349
\vrule 69–70, *178*, 281, 349
  horizontal par nature 82
\vsize 63, 77, *146*, 282, 283, 349
  déterminé par \magnification 231
\vskip 93, *161*, 349
\vsplit 151, *155*, 349
\vss *164*, 278, 349
\vtop 51, 52–53, 100, *167*, 349

\wd *173*, 349
\wedge *197*, 349
\widehat *207*, 349
\widetilde *207*, 349
\widowpenalty *144*, 349
\wlog *269*, 350
  développé par les règles d'\edef 239
\wp *196*, 350
\wr *197*, 350
\write *258*, 350
  avec \immediate 258
  développé durant \shipout 154
  développé par les règles d'\edef 239
  écrire % avec 302
  élément extraordinaire produit par 66
  expansion de \'*c* en 84
  flot d'entrée pour 68
<write> 298
\writetocentry *311*

\xdef 71, 236, *239*, 350
\xi *195*, 350
\Xi *195*, 350
\xleaders 91–93, *179*, 350
\xrdef *312*
\xref *312*
\xrefn *312*
\xspaceskip *113*, 350

\year *233*, 307, 350
yeux 16, *48*
  Voir aussi Anatomie de TEX

Zapf, Hermann 36
\zeta *195*, 350

à propos des auteurs

Paul W. Abrahams, Sc.D., CCP, est consultant informatique et ancien président de l'Association for Computing Machinery. Ses spécialités sont la programmation de langage informatique, la conception et implémentation de systèmes logiciels et l'écriture technique. Il a reçu son doctorat de mathématiques du Massachusetts Institute of Technology en 1963, étudiant l'intelligence artificielle sous la tutelle de Marvin Minsky et John McCarthy. Il est un des concepteurs du premier système LISP et concepteur du système CIMS PL/I, développé quand il était professeur à l'université de New York. Plus récemment, il a conçu SPLASH, un langage de programmation système pour des hackers de logiciel. Paul réside à Deerfield, Massachusetts, où il écrit, bidouille, fait de la randonné, chasses les champignons sauvages et écoute de la musique classique.

Kathryn A. Hargreaves a reçu son M.S. degree en informatique de l'université du Massachusetts, Boston, en août 1989. Ses spécialités sont la typographie numérique et la vision humaine. Elle a développé un ensemble de programmes pour produire une police numérique de haute qualité et librement distribuable pour la Free Software Foundation et a également travaillé avec Robert A. Morris en tant qu'Adjunct Research Associate. En 1986 elle a accompli le programme de ré-entrée en informatique pour les femmes et les minorités à l'université de Californie à Berkeley, où elle a également travaillé dans le TEX research group sous Michael Harrison. Elle a étudié la conception de police de caractère avec Don Adleta, André Gürtler, et Christian Mengelt à la Rhode Island School of Design. Typographe, elle a travaillé chez Headliners/ Identicolor, à San Francisco, et chez Futur Studio, Los Angeles, deux sociétés typographiques majeures. Elle a obtenu également un M.F.A. en Peinture/Sculpture/Art Graphiques de l'université de Californie à Los Angeles. Kathy peint des aquarelles, conçoit des polices, joue du piano et lit la critique de film féministe.

Comme Kathy, Karl Berry a reçu son M.S. degree en informatique à l'université du Massachusetts, Boston, en août 1989. Il a également travaillé pour la Free Software Foundation, a fait de la recherche avec Morris et a étudié avec Adleta, Gürtler, et Mengelt. Il avait commencé à travailler avec TEX depuis 1983 et a installé et a maintenu le système TEX dans un certain nombre d'universités. Il était le mainteneur du système Web2c développé par Tim Morgan pendant un certain nombre d'années, parmi d'autres projets TEX. Il est devenu président du TEX Users Group en 2003.

Colophon

Ce livre a été composé en utilisant TEX (naturellement), développé par Donald E. Knuth. Le texte principal est composé en Computer Modern, également conçu par Knuth. Les en-têtes du livre original ont été composées en Zapf Humanist (la version Bitstream d'Optima), conçu par Hermann Zapf.

Le papier était du Amherst Ultra Matte 45 livres. L'impression et la reliure ont été faits par Arcadia Graphics-Halliday. La sortie de photocomposeuse a été produite à type 2000, Inc., à Mill Valley, Californie. Les preuves ont été faites sur une Apple LaserWriter plus et sur une Hewlett Packard LaserJet II.

Les références croisées, l'indexage et la table des matières ont été faits mécaniquement, en utilisant les macros de la section 12 ainsi que des macros additionnelles écrites spécialement pour ce livre. La production de l'index a été réalisée par un programme additionnel écrit en Icon.

Liste des concepts

alignement 45
Anatomie de TEX 48
argument 49
ASCII 50
assignement 51
boîtes 51
caractère 53
caractère actif 54
caractère d'échappement 55
césure 55
classe 56
codes de catégorie 56
commande 58
constantes décimales 59
construction de page 59
coupure de ligne 59
coupure de page 60
crénage 61
dimension 61
disposition de page 62
délimiteur 63
démérites 65
élément 65
élément extraordinaire 66
entête 66
espace 66
étirement 67
famille 67
fichier 67
fichier format 69
fichier log 69
filets 69
flots d'entrée 70
flots de sortie 70
global 70
groupe 71
hauteur 71
hbox 71
insertion 72
largeur 72
ligature 72
ligne de base 72
liste 73
liste horizontale 73
liste verticale 73
macro 73
magnification 76
marges 77
marque 77
mathcode 78
mathématique affichée 79
médiocrité 79
mode 80
mode horizontal 81
mode mathématique 81
mode ordinaire 82
mode restreint 82
mode vertical 82
mot de contrôle 83
muglue 83
nom de fichier 83
nombre 83
outer 84
page 85
paragraphe 86
paramètre 87
pénalité 87
pied de page 88
plain TEX 88
point de référence 88
police 88
primitive 90
profondeur 90
registre 90
réglures 91
ressort 93
ressort inter-ligne 95
rétrécissement 95
routine de sortie 96
séquence de contrôle 96
strut 97
style 98
symbole de contrôle 98
taille script 98
taille scriptscript 99
taille texte 99
test conditionnel 99
TEX MEX 99
texte justifié 99
texte mathématique 100
token 100
unité de mesure 100
unité mathématique 100
vbox 100