Guide LuaLaTeX

Stéphane Vinatier
stephane.vinatier@unilim.fr

2024

Nous présentons ici ce qui est sans doute la version la plus moderne de TeX. Appelée LuaTeX (ou LuaLaTeX pour sa variante qui contient LaTeX), elle permet notamment d'utiliser, dans un document TeX (ou LaTeX) :

- tous les caractères encodés en `utf-8`, la norme actuelle de l'encodage ;
- toutes les fontes de caractères OpenType et TrueType, les standards actuels pour les fontes ;
- le langage de programmation `lua`, facile à utiliser et à interfacer avec d'autres langages ;
- les nombreuses bibliothèques de `lua` (notamment celles qui gèrent les fontes).

Dans ce document, nous nous concentrons sur les outils de sélection de fontes de LuaLaTeX et sur l'utilisation de `lua` pour programmer à l'intérieur d'un document LaTeX.

Table des matières

1 Encodage des caractères

Pour passer du fichier source `guide-book.tex` au fichier mis en pages `guide-book.pdf`, il faut utiliser un *moteur* capable de lire le code source et d'interpréter tout ce qui s'y trouve : caractères et commandes (aussi appelées macros ou séquences de contrôle), et de composer un contenu mis en

page à partir de cela. Le moteur le plus usuel est `pdflatex`, du nom de l'instruction qu'on utiliserait en ligne de commande si on voulait compiler le *Guide* LaTeX– TikZ– `pgfplots` depuis un terminal :

```
pdflatex guide-book.tex
```

La plupart du temps, l'instruction est lancée directement, de façon assez transparente, par l'éditeur de texte dédié à LaTeX (éventuellement `overleaf`) lorsqu'on clique sur le bouton « compilation ». En fait, `pdflatex` est une simple variante, intégrant le format LaTeX, du moteur `pdftex`. Tous deux ont été développés en intégrant le moteur d'origine, le plus basique, `tex` (qui a lui aussi une variante `latex` pour LaTeX).

Encodage `ASCII`. Le moteur `tex` a été conçu à l'époque (les années 1970-80) où l'encodage des caractères le plus fréquemment utilisé était l'`ASCII`, qui encode les caractères sur 7 bits (avec seulement $2^7 = 128$ caractères possibles). Il fallait alors, pour faire afficher une lettre accentuée, utiliser une commande spécifique, par exemple `\`a` pour à. À cette époque aussi, plusieurs extensions de l'ASCII sont apparues, encodant les caractères sur 8 bits (soit 1 octet, 256 caractères disponibles), notamment l'encodage `latin-1` (norme ISO 8859-1), pour inclure la plupart des caractères européens (dont les lettres accentuées, æ, les cédilles,...). Il n'a pas été difficile d'adapter TeX et LaTeX à ces nouveaux encodages car les moteurs associés, `tex` et `latex`, lisent justement le code source octet par octet. Les moteurs `pdftex` et `pdflatex`, qui contrairement aux précédents produisent directement le fichier `pdf` à partir du code source, lisent eux aussi le code source octet par octet.

Encodage `utf-8`. Est arrivé ensuite l'encodage `utf-8`, extension de `latin-1` qui utilise jusqu'à 4 octets par caractère (entre 1 et 4 octets, les premiers bits indiquant le nombre d'octets à lire pour retrouver l'encodage complet). Il utilise le standard `unicode`, qui répertorie actuellement 149 813 caractères [1] en leur attribuant un numéro appelé *point de code*, ce qui permet d'encoder virtuellement tous les caractères de tous les alphabets du monde et de toutes les époques connues, en plus d'un grand nombre de symboles d'usage plus ou moins courant. Il existe d'autres manières d'encoder les caractères `unicode` (notamment `utf-16` et `utf-32`), cependant `utf-8` est le standard le plus majoritairement utilisé [2].

Il est bien sûr possible, pour un jeu de caractères encodés en `utf-8` (par exemple les caractères les plus courants, ou ceux de l'alphabet d'une langue donnée), de créer des commandes qui, à partir de la suite d'octets correspondant à un caractère, affichent le signe correspondant (de la même manière que `\`a`, qui occupe 3 octets, affiche à). Ces commandes sont alors incluses dans un package, par exemple `inputenc` avec l'option `utf8` [3], qu'on appelle dans le préambule et qui permet d'utiliser simplement les caractères en question.

Deux nouveaux moteurs. Ce serait une autre paire de manches de créer des commandes qui affichent convenablement les 150 000 caractères `utf-8` possibles et ce ne serait pas la solution la plus efficace. C'est pourquoi deux nouveaux moteurs, nommés `xetex` et `luatex`, ont été développés depuis une quinzaine d'années (avec des variantes pour LaTeX), qui lisent directement les caractères

1. Dans la version 15.1.0, au 23 septembre 2023, voir https://www.unicode.org/versions/Unicode15.1.0/
2. Voir https://fr.wikipedia.org/wiki/UTF-8
3. Ce package est en fait inclus dans les distributions récentes de LaTeX, si bien qu'il n'est plus nécessaire de le charger dans le préambule.

du code source en utf-8, sans besoin d'aucun package spécifique pour cela : ils incluent nativement la compréhension de l'encodage des caractères utf-8.

Le code source d'un document peut désormais contenir des caractères de toutes les langues du monde et les symboles les plus variées, les moteurs xetex et luatex n'auront aucune difficulté à les interpréter. Par exemple, dans ce document produit avec LuaLaTeX, on a placé (en le copiant-collant depuis un autre fichier pdf) le caractère

✎

(oui, c'est un caractère utf-8 !) directement dans le code source (où l'éditeur utilisé le fait apparaître tel quel). Le moteur utilisé, lualatex, l'a interprété et a inséré le dessin qui le représente dans le fichier pdf produit comme il le fait pour n'importe quel autre caractère utf-8, lettre, chiffre, symbole de ponctuation ou autre…

Le choix de la fonte. Attention cependant, le fait que le moteur utilisé interprète correctement le caractère ne signifie pas qu'il sera nécessairement affiché comme on le souhaite dans le fichier pdf produit. Pour cela, il faut que le dessin du caractère, son *glyphe*, soit accessible au moteur de compilation, si possible dans la fonte utilisée à l'endroit du document où il apparaît. Si la fonte en question ne contient pas de glyphe pour le caractère, celui-ci ne pourra pas être affiché dans le pdf (dans certains cas, lualatex modifie la fonte au moment de la compilation pour pouvoir l'afficher tout de même, et inscrit un message dans le fichier de compilation). Or aucune fonte ne contient des glyphes pour tous les caractères utf-8 possibles [4].

Il est donc important de pouvoir sélectionner facilement la fonte utilisée dans un document et, le cas échéant, de pouvoir la modifier pour certains caractères particuliers. Dans l'exemple ci-dessus, il a fallu sélectionner la fonte DejaVu Sans pour faire afficher le glyphe du crayon. Justement, XeLaTeX et LuaTeX, ainsi bien sûr que leurs variantes pour LaTeX, permettent un choix très facile et très large de la fonte qu'on utilise, à l'aide du package fontspec (qui leur est commun). Nous allons détailler ce point dans la partie suivante.

2 Gestion des fontes

2.1 Le préambule

Il y a peu de modifications à apporter au préambule d'un fichier prévu pour le moteur latex pour qu'il puisse bénéficier des avantages du moteur lualatex en matière de gestion des fontes : on retire les instructions concernant l'encodage des caractères et les fontes (\usepackage[utf8]{inputenc} [5], \usepackage[T1]{fontenc} et \usepackage{lmodern}) et on les remplace par le chargement du package fontspec et le choix d'une fonte principale pour le document :

```
\documentclass{article}
```

4. D'après [And15, p. 12-13], rédigé en 2015, les fontes les plus fournies en caractères sont *Arial Unicode MS* avec 50 000 glyphes et *Bitstream Cyberbit* avec 30 000 glyphes — on est encore loin des 150 000 caractères d'unicode. Voir aussi https://en.wikipedia.org/wiki/Unicode_font#List_of_Unicode_fonts

5. Comme indiqué plus haut, ce package est désormais inclus dans les distributions de LaTeX et peut donc être omis même pour une compilation avec pdflatex.

```
\usepackage{fontspec}
\setmainfont{Linux Libertine O}

\usepackage[a4paper]{geometry}
\usepackage[french]{babel}

\usepackage{amsfonts,amssymb,amsmath,amsthm}
\usepackage{enumerate}
\usepackage{graphicx}
\usepackage{array}

\usepackage{hyperref}
\hypersetup{pdfstartview=XYZ}

\begin{document}
...
\end{document}
```

Attention, le fichier obtenu n'est plus compilable avec `pdflatex` qui ne connaît pas le package `fontspec` (réciproquement, la présence des packages `inputenc` ou `fontenc` dans un fichier compilable par `pdflatex` risque de le rendre non compilable par `lualatex`).

Les dénominations `fontspec` et `\setmainfont` font référence à un choix de *fonte*, c'est-à-dire un « ensemble des caractères de la même famille (par exemple le *Garamond*) »[6], ce qu'on appelle parfois aussi *police*. Une fonte donnée contiendra souvent en différentes tailles à la fois des caractères romains (droits), *italiques* (penchés), **gras** (épais) ; certaines proposent aussi du *slanted*[7] (commande `\textsl{..}`), du ***gras-italique*** (commande `\textit{\bfseries ..}`), plusieurs variétés de **gras** (plus ou moins épais), des Petites Capitales (commande `\textsc{..}`),... On verra en §2.4 comment suppléer certains manques en utilisant les fonctionnalités de `fontspec`.

La fonte principale sélectionnée par la commande `\setmainfont` est souvent choisie avec empattement (serif). On peut également, si cela est nécessaire (par exemple pour certains titres ou pour écrire du code informatique), spécifier des fontes sans serif et à chasse fixe avec les commandes `\setsansfont{...}` et `\setmonofont{...}`, à placer également dans le préambule.

2.2 Choix des fontes

Fonte TEXLive. On peut continuer à utiliser les fontes incluses dans TEXLive, en les appelant par leur nom « humain » (*Latin Modern Roman* à la place de `lmodern`), par exemple les suivantes :

6. https://www.cairn.info/revue-document-numerique-2006-3-page-7.htm
7. Ici, le *slanted* semble être identique à l'*italique*, ce qui signifie que la fonte choisie, *Linux Libertine O*, ne contient pas cette variante de caractères penchés.

Nom de la fonte	romain	*italique*	**gras**
Latin Modern Roman	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
Noto Serif	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
TeX Gyre Pagella	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
TeX Gyre Termes	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
TeX Gyre Heros	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
TeX Gyre Bonum	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**

Les fontes *Tex Gyre Pagella*, *Termes*, *Heros* et *Bonum* sont inspirées des fontes « classiques » Palladio, Times Roman, Helvetica et Bookman.

Citons aussi les fontes *DejaVu* avec notamment DejaVu Serif, DejaVu Sans et DejaVu Sans Mono, dont on voit qu'elles utilisent des caractères plus grands que les autres (c'est pourquoi elles ne figurent pas dans la table ci-dessus, qu'elles feraient déborder dans la marge). Une des fonctionnalités avantageuses de `fontspec` est qu'on peut, au moment de la sélection de la fonte, appliquer une option d'échelle pour modifier les dimensions de ses caractères. Avec l'instruction

```
\setmainfont[Scale=.8]{DejaVu Serif}
```

dans le prémbule, on obtient la réduction de la fonte DejaVu Serif à une taille comparable à celle de la fonte courante : DejaVu Serif.

Fontes disponibles sur l'ordinateur. Toute fonte disponible sur l'ordinateur est utilisable : un moyen simple d'en trouver est de les faire défiler dans le menu de choix de fonte d'un logiciel comme *Libre Office Writer*. Voici quelques exemples :

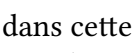
Nom de la fonte	romain	*italique*	**gras**
C059	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
Caladea	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
Free Serif	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
Liberation Serif	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
Montserrat	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
Nimbus Roman	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
Old Standard	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
P052	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
STIX	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**
URW Bookman	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**

Certaines d'entre elles existent aussi en version *Sans*, *Mono* et autres variantes. Pour voir *toutes* les fontes du système, taper `fc-list` en ligne de commande sous PC ou utiliser l'application *Fontbook* sous MAC OS X [Pég10b, p. 16-17].

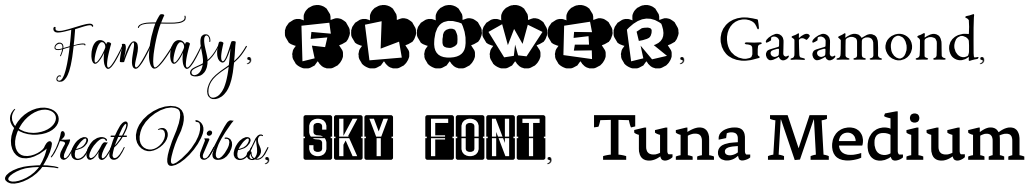
Télécharger une fonte. Enfin on peut récupérer une fonte sur internet (gratuite ou payante), l'installer sur son ordinateur et l'utiliser aussi simplement que les autres. Par exemple, l'article [BA22] présente la fonte Infini, qui est téléchargeable gratuitement sur https://www.cnap.fr/sites/infini/en/ et vient avec un mode d'emploi (succint) pour l'installation.

Nom de la fonte	romain	*italique*	**gras**
Infini	ABC abc 012...89	*ABC abc 012...89*	**ABC abc 012...89**

Elle propose aussi des majuscules ornées (fonte *Infini-picto* avec l'option `Scale=1.5`) :

Le codage `\LaTeX` donne, dans cette fonte :  .

L'article [And15] propose de nombreuses pistes pour télécharger d'autres fontes. On pourra passer beaucoup de temps à tester les très nombreuses fontes proposées sur différents sites web, malheureusement souvent de manière partielle contrairement à la fonte Infini présentée ci-dessus (parfois pas de chiffres, ou pas de minuscules, pas d'italique…). Quelques exemples :



2.3 Changement ponctuel de fonte

On peut modifier ponctuellement la fonte utilisée, comme cela a déjà été fait à de nombreuses reprises dans ce document, en utilisant la commande `\fontspec` (du package `fontspec`!). Par exemple

```
{\fontspec{Infini} Vers l'infini...} dans la fonte \textit{Infini}.
```

affiche : Vers l'infini… dans la fonte *Infini*. Les accolades ouvrante `'{'` et fermante `'}'` qui encadrent la commande et le texte l'enserrent dans un *groupe*, structure de base de TEX qui limite la portée de la commande à l'intérieur du groupe. La fonte principale est donc de nouveau en usage après l'accolade fermante.

Un usage particulier du changement ponctuel de fonte est l'insertion d'une *lettrine*, lettre de grande taille, ornée ou non, placée au début d'un chapitre ou d'un paragraphe comme c'est le cas pour celui-ci. Une fois chargé le package `lettrine` dans le préambule, les commandes utilisées sont :

```
\renewcommand*{\LettrineFontHook}%
              {\fontspec{GreatVibes-Regular.otf}}
\lettrine[lines=2,loversize=0.5]{U}{\ \ n usage} particulier
```

On sélectionne la fonte désirée en redéfinissant la commande `\LettrineFontHook` (propre au package `lettrine`), ici il s'agit d'une fonte téléchargée sur internet et simplement déposée dans le répertoire courant (mais pas installée dans le système de fontes). Ensuite la commande `\lettrine` permet de sélectionner sa taille en nombre de lignes, de combien elle peut dépasser au-dessus du paragraphe et quelles lettres sont écrites en petites capitales dans ce qui suit. Noter le « bricolage », sans doute pas optimal, avec les espaces `\ \` ajoutés avant le 'n' pour le décaler vers la droite, vu la forme de la lettrine U.

Caractères spéciaux. Certains caractères `utf-8` ne sont disponibles que dans certaines fontes. En réalité, en dehors des caractères européens déjà inclus dans l'encodage `ASCII` ou `latin-1`, c'est le cas de la plupart des caractères disponibles en `utf-8`, par exemple ceux des alphabets non européens ou des symboles d'usage local. L'article [And15] prend en exemple le symbole de la monnaie de l'Azerbaïdjan, le *manat*. Son point de code, ou numéro dans la norme `unicode`, est `U+20BC` (en écriture hexadécimale), si bien qu'à défaut de le copier-coller depuis un autre document (pdf ou internet), on peut demander son affichage avec la commande

```
\char"20BC
```

Attention à la double apostrophe qui n'est pas forcément celle qu'on obtient directement avec la touche correspondante du clavier (souvent celle du 3) : en effet, dans certains éditeurs dédiés à LaTeX, cette touche produit deux apostrophes simples au lieu d'une double (deux signes au lieu d'un donc) ; le résultat peut paraître identique à première vue mais ne l'est pas (`\char''20BC` au lieu de `\char"20BC`). La « bonne » double apostrophe s'obtient (sur mon clavier) en appuyant *deux fois* consécutives sur la touche de l'apostrophe double.

Cependant la fonte principale utilisée dans ce document ne contient pas de signe de numéro `U+20BC`, ce qui produit l'affichage ⍰ dans le texte à la place du symbole voulu (ce caractère de remplacement dépend de la fonte courante) et un message d'erreur à la compilation :

```
Missing character: There is no ₼ (U+20BC) in font LinuxLibertineO:mode=node;s
cript=latn;language=dflt;+tlig;!
```

Il faut donc trouver une fonte qui contienne le signe approprié pour ce caractère, c'est le cas par exemple si l'on choisit :

```
{\fontspec{Symbola}\char"20BC}
```

On obtient en effet que le symbole du manat est ₼.

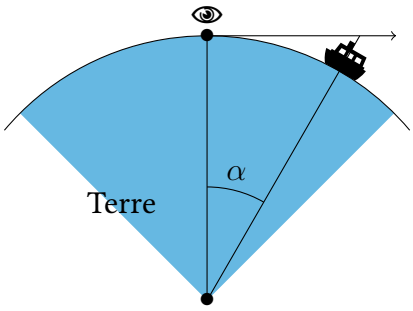
Le symbole de l'euro dans la fonte courante est € (caractère inséré dans le code source par la combinaison « `alt gr + e` », ou avec `\char"20AC`), alors que dans la fonte *Symbola* on obtient €. L'article [And15, p. 5] indique où trouver cette fonte et comment savoir, pour un caractère donné, quelles fontes sont capables de l'afficher.

Par ailleurs, il donne aussi en fin d'article un fichier LuaLaTeX permettant d'afficher tous les caractères disponibles d'une fonte donnée. En l'appliquant à la fonte courante *Linux Libertine O*, jusqu'au numéro `FFFF`, on trouve quelques caractères inattendus :

Noter la *ligature* élégante Qu (produite ici par la commande `\char"E048`) que la fonte *Linux Libertine O* substitue aux lettres Q et u (elle apparaît aussi dans l'Exercice 1 ci-dessous, où l'on a simplement écrit Q suivi de u).

Exercice 1. Que représente le caractère de point de code `U+270E` ?

Exercice 2. Reproduire le dessin suivant :



Indication : on pourra lancer une recherche sur internet avec les mots : « bateau unicode » et « œil unicode », pour trouver les points de code des caractères apparaissant sur le dessin. On pourra utiliser la fonte Symbola *pour les afficher.*

Caractères privés. La norme unicode offre des plages réservées à des usages privés, c'est-à-dire des intervalles de points de code qui ne sont pas attribués à des caractères particuliers et qu'une fonte pourra utiliser pour encoder des caractères qui lui sont propres. L'article [And15, §4.2] donne un exemple d'utilisation d'une plage privée pour des caractères médiévaux [8].

Un autre exemple est donné dans [BA22, p. 71-78], pour la fonte *Infini* principalement mais aussi pour quelques autres, en décrivant comment les fonctionnalités de fontspec, dont nous reparlons ci-après, permettent d'accéder "facilement" à ces caractères privés. Comme pour la ligature Qu de *Linux Libertine O*, sont concernées notamment les nombreuses ligatures originales de lettres capitales proposées par *Infini* :

Q, ASE, E, TURE, TY, UN, ON, LE, ER, RE, D, ME, …

Voir la belle Figure 8 de [BA22, p. 44] pour encore plus de ligatures. De même les caractères « acrophoniques » de la fonte *Infini-picto*, A B C D E…, utilisent des points de code de plages privées d'unicode. Les ligatures classiques (ff, fi, ffi, fj, ffj, fl, ffl) ont droit, elles, à un point de code spécifique dans unicode.

2.4 Un coup d'œil sur les options de fontspec

On ne fait ici qu'effleurer la description des très nombreuses options de fontspec, dont les possibilités sont extrêmement riches et variées, on renvoie au manuel [Rob24] pour des détails. En particulier, on y verra que ces possibilités dépendent à la fois du moteur utilisé (xetex ou luatex) et du type de fonte choisi (essentiellement TrueType — nom de fichier en .ttf — ou OpenType — nom de fichier en .otf). Les deux fonctionnalités décrites ci-dessous sont toujours disponibles.

On a vu plus haut l'option Scale=.8 permettant de réduire légèrement la taille de la fonte *DejaVu Serif* de 80%. Il existe d'autres façons de régler la taille d'une fonte. Dans cet exemple, comme l'objectif est de la rendre de même taille que la fonte courante, la solution la plus efficace est la suivante :

```
\scalebox{1.3}{Passer de la fonte Linux Libertine O
```

8. Il présente aussi des fontes de symboles ou d'ornement remplaçant les caractères habituels — à commencer par les lettres — par des signes ou des décorations.

```
  {\fontspec[Scale=MatchLowercase]{DejaVu Serif}à la fonte DejaVu
Serif,}
  et retour.}
```

Passer de la fonte Linux Libertine O à la fonte DejaVu Serif, et retour.

Les valeurs MatchUppercase et MatchAveragecase existent aussi pour cette option, voir le manuel de fontspec [Rob24, p. 27].

Ce manuel propose aussi des options permettant de suppléer l'absence dans une fonte de certains types de caractères. Par exemple, on a remarqué que la fonte courante de ce document, *Linux Libertine O*, ne dispose pas de caractères *slanted* (penchés). Les commandes

```
\textit{Comparons les caractères en italique et ceux en slanted.}\par
\textsl{Comparons les caractères en italique et ceux en slanted.}
```

donnent des résultats parfaitement identiques :

Comparons les caractères en italique et ceux en slanted.

Comparons les caractères en italique et ceux en slanted.

Faisons précéder la deuxième ligne de l'instruction \addfontfeature{AutoFakeSlant}, on obtient alors :

Comparons les caractères en italique et ceux en slanted.

à comparer avec le romain (droit) :

Comparons les caractères en italique et ceux en slanted.

C'est bien du *romain penché* qu'on a fabriqué !

Cette fonctionnalité, qu'on peut également appliquer au gras, peut aussi être placée en option de la commande \setmainfont, qui déclare la fonte principale (dans le préambule si cela porte sur tout le document ou dans le texte, à l'intérieur d'un groupe, si cela ne concerne que ce groupe). On en trouve un exemple dans [BA22, p. 54], où ce procédé est utilisé pour obtenir du gras-penché suppléant l'absence de ***gras-italique*** dans la fonte Infini :

```
  {\setmainfont{Infini}[
      Scale=MatchLowercase,
      BoldItalicFont = *-Bold,
      BoldItalicFeatures={FakeSlant=0.087}]
    Voici un texte (...) en {\itshape\bfseries gras-penché}, (...).}
  Retour à la fonte \textit{Linux Libertine O}, (...)
```

Voici un texte écrit avec la fonte *Infini* en romain, en **gras**, en *italique*, en ***gras-penché***, où l'on voit que ce dernier n'est pas de l'italique graissé (en particulier sur la lettre « a »). Retour à la fonte *Linux Libertine O*, suite à l'accolade fermante '}' qui referme le groupe contenant la « nouvelle » commande \setmainfont (l'accolade ouvrante '{' était placée juste avant cette commande).

Par rapport à l'exemple donné dans [BA22], on a ajouté l'option `Scale=MatchLowercase` à la commande `\setmainfont` de façon à ce que la hauteur des caractères de la nouvelle fonte s'ajuste à celle de la précédente (les caractères de la fonte Infini sont par défaut plus hauts que ceux de la fonte courante). On aurait pu aussi utiliser la syntaxe :

```
{\fontspec[Scale=MatchLowercase,BoldItalicFont = *-Bold,
BoldItalicFeatures={FakeSlant=0.087}]{Infini}Voici un texte ... }
```

qui produit le même résultat.

3 Programmation avec `lua`

Un autre apport essentiel de LuaLaTeX est la possibilité de faire interagir des programmes en `lua` avec le code LaTeX. Pour cela il y a deux commandes de base (qui ne demandent aucun package spécifique, seulement la compilation avec le moteur `luatex` ou `lualatex`) :

— `\directlua{ ... }` pour inclure du code `lua` dans le code source LaTeX ;
— `tex.sprint( ... )` pour faire écrire `lua` dans le code compilé par LuaLaTeX.

En fait `lua` n'écrit pas directement dans le code source, il ne modifie pas celui-ci ; cependant tout se passe à la compilation comme si les morceaux de code LaTeX passés par `tex.sprint` y étaient écrits. L'utilisation de cette instruction crée donc un code source « virtuel », non visible par l'utilisateur mais qui est lu par le moteur `lualatex`. Par exemple, la commande

```
$\pi =\directlua{tex.sprint(math.pi)}$
```

passe à `lua` l'instruction `tex.sprint(math.pi)`, c'est-à-dire demande à `lua` d'écrire dans le code source virtuel la valeur de la constante mathématique `math.pi` ; le moteur de compilation lit donc `\pi=3.1415926535898` et affiche $\pi = 3.1415926535898$.

On note que, conformément aux habitudes, la commande LaTeX `\directlua` commence par le caractère d'échappement (antislash \) et reçoit son argument entre accolades `{...}`, tandis que l'instruction `lua` qu'elle contient, `tex.sprint`, n'est pas précédée d'une antislash et reçoit son argument entre parenthèses `(...)`. Ce qui est contenu dans l'argument de `\directlua`, entre les accolades, est lu par LaTeX avant d'être envoyé à l'interpréteur de `lua`. La manière dont LaTeX lit du code et le « digère », avant de le passer à `lua`, peut causer quelques surprises. Par exemple, [Pég10a, p. 19] signale que LaTeX remplace toutes les fins de lignes par des espaces, si bien que `lua` ne voit qu'une seule ligne d'instructions, dans laquelle il ne faut donc pas insérer le signe '−−', qui introduit les commentaires pour `lua`, faute de quoi tout ce qui suit (même sur les "lignes" suivantes dans le code d'origine), sera ignoré.

3.1 Le package `luatextra`

Ce genre de chausse-trappe est en partie évitable en chargeant le package `luacode` dans le préambule, ou le package `\luatextra` qui le contient [9]. Il donne accès à la comande `\luadirect {...}` qui se sustitue à `\directlua` et traite certains de ces problèmes. On verra dans [Pég10a,

9. [Pég10a, p. 26] indique que ce package « charge les extensions usuelles pour LuaTEX, actuellement `fontspec`, `luacode`, `metalogo` (macros pour obtenir les différents logos, en particulier LuaTeX et LuaLaTeX), `luatexbase`, `lualibs`, `fixltx2e` (correctifs et améliorations du noyau de LaTeX) ».

p. 2] des variantes de cette commande ainsi que les façons de passer certains caractères spéciaux de TEX en argument, en fonction de la variante choisie (les caractères _ , ^, & , $, { et } passent sans problèmes, seuls \ , # , ~ et % nécessitent des précautions).

Le package `luacode` fournit aussi un environnement `luacode`, c'est-à-dire des balises **\begin**`{luacode} ...` **\end**`{luacode}` entre lesquelles on peut insérer le code `lua` auquel on veut faire appel dans le document. Cependant, le manuel de `luacode` [Pég12] tout comme [Pég10a, p. 21] recommandent de privilégier l'appel à un fichier externe, contenant le code `lua`, dès lors que ce code n'est pas « trivial » comme dans l'exemple ci-dessus.

Un préambule "standard" pour un document LuaLATEX utilisant les possibilités d'interaction avec le langage `lua` pourrait donc être le suivant :

```
\documentclass{article}

\usepackage{luatextra}
\setmainfont{Linux Libertine O}

\usepackage[a4paper]{geometry}
\usepackage[french]{babel}

\usepackage{amsfonts,amssymb,amsmath,amsthm}
\usepackage{enumerate}
\usepackage{graphicx}
\usepackage{array}

\usepackage{hyperref}
\hypersetup{pdfstartview=XYZ}

\begin{document}
...
\end{document}
```

3.2 Appel à un fichier `lua` externe

Les tableaux de démonstration de fontes de la partie 2.2 sont produits automatiquement à l'aide d'une fonction écrite en `lua`, nommée `luapoliste` et contenue dans le fichier `polices.lua`, prenant en arguments un texte de démonstration et une liste de noms de fontes :

```
\luadirect { dofile( 'polices.lua' ) }%
\gdef\poliste#1#2{\luadirect {luapoliste(\luastring{#1},\luastring
{#2})}}%
\poliste{ABC abc 012...89}{Latin Modern Roman, Noto Serif, TeX Gyre
Pagella, TeX Gyre Termes, TeX Gyre Heros, TeX Gyre Bonum}
```

- Comme on le voit, l'appel au fichier `polices.lua` se fait via l'instruction `dofile( 'polices.lua' )` passée à `lua` à l'aide de la commande `\luadirect`.

- Cet appel étant fait, les fonctions lua définies dans le fichier sont accessibles, en particulier celle nommée luapoliste, à laquelle la nouvelle commande LaTeX nommée \poliste fait appel via \luadirect (en lui « passant » ses deux arguments #1 et #2).
- On a utilisé la *primitive* TeX **\gdef** plutôt que la commande LaTeX **\newcommand** car le bloc ci-dessus est contenu dans un groupe (environnement **\begin**{center} ... **\end**{center}), ce qui limite normalement la portée de la définition de la commande à l'intérieur de ce groupe ; or cette commande est appelée à resservir plus loin. La définition avec **\gdef**, le 'g' initial signifiant *global*, permet à cette commande d'être utilisable en dehors du groupe où elle est définie.
- On peut alors utiliser la commande LuaLaTeX \poliste avec les arguments que l'on souhaite. À sa lecture, le moteur lualatex insèrera à sa place dans le code source virtuel les lignes de code LaTeX produites par la fonction luapoliste appliquée aux arguments choisis.

Chaînes de caractères et développement de macros. La commande \luastring qui entoure les arguments #1 et #2 dans la fonction luapoliste est le raccourci fourni par le package luacode de la commande LuaLaTeX au nom peu pratique \luatexluaescapestring. Elle permet de passer les arguments tels quels, sous forme de chaînes de caractères, en évitant les problèmes posés par certains caractères.

Elle admet deux variantes, respectivement \luastringN et \luastring0, dans lesquelles les macros LaTeX éventuellement contenues dans les arguments ne sont pas développées (c'est-à-dire ne sont pas remplacées par leurs textes de remplacement), respectivement sont développées une seule fois. Avec \luastring, elles sont entièrement développées (c'est-à-dire les macros qui apparaissent à un stade quelconque du développement sont à leur tour remplacées par leur texte de remplacement, jusqu'à ce qu'il n'y ait plus aucune macro).

Déboggage. Noter également la possibilité, avec la commande \LuaCodeDebugOn du package luacode, de faire écrire le code reçu par lua dans le fichier .log produit à la compilation, à partir de cette commande et jusqu'à la commande \LuaCodeDebugOff suivante. En plaçant ces deux commandes autour du premier appel à la fonction luapoliste, au début de la partie 2.2, on obtient dans le fichier .log :

```
-- BEGIN luacode debug (on input line 262)
 dofile( 'polices.lua' )
-- END luacode debug (on input line 262)
-- BEGIN luacode debug (on input line 265)
luapoliste("ABC abc 012...89","Latin Modern Roman, Noto Serif, TeX
Gyre Pagella, TeX Gyre Termes, TeX Gyre Heros, TeX Gyre Bonum")
-- END luacode debug (on input line 265)
```

On constate ainsi que lua reçoit correctement les chaînes de caractères en argument de la fonction luapoliste.

3.3 Une parenthèse sur les macros

Une *macro* est une commande définie dans le code source par l'utilisateur, soit à l'aide d'une primitive TeX (**\def** ou ses variantes), soit à l'aide de la commande LaTeX **\newcommand** (ou sa variante **\newcommand***). Définir une macro consiste à indiquer au moteur de compilation le *texte de*

remplacement qu'il devra utiliser lorsqu'il rencontrera le *nom* de la macro dans la suite du code. Ce texte est tout ce qui se trouve entre l'accolade ouvrante `'{'` et l'accolade fermante `'}'` [10] qui suivent l'une des séquences **\def**⟨*nom*⟩ ou **\newcommand**{⟨*nom*⟩} (les accolades autour du nom de la macro sont facultatives). Si la macro admet des arguments (au plus 9), ceux-ci sont annoncés entre le nom de la macro et le texte de remplacement, avec des syntaxes légèrement différentes :

- avec **\def**, les arguments sont indiqués les uns à la suite des autres, avec des # numérotés de 1 en 1 dans l'ordre à partir de 1, par exemple `#1#2#3#4#5` pour une commande à 5 arguments ;
- avec **\newcommand**, on indique simplement le nombre d'arguments entre crochets, là aussi entre le nom de la macro et le texte de remplacement, par exemple :

```
\newcommand\poliste[2]{\luadirect {luapoliste(\luastring{#1},
\luastring{#2})}}
```

Dans cet exemple, le texte de remplacement est `\luadirect {luapoliste(\luastring{#1}, \luastring{#2})}`, c'est ce que le moteur enregistre lorsqu'il rencontre la définition de la macro. Ensuite, lorsque la commande `\poliste{ABC abc 012...89}{Infini}` apparaît dans le code source, il substitue `ABC abc 012...89` à `#1` et `Infini` à `#2` dans le texte de remplacement, ce qui donne :

```
\luadirect{luapoliste(\luastring{ABC abc 012...89},\luastring{Infini})
}
```

Ce nouveau code est lui-même une macro, constituée de la commande `\luadirect` qui envoie son argument dans `lua`, après lecture de celui-ci par le moteur de compilation, qui passe donc `ABC abc 012...89` et `Infini` à `lua` sous forme de chaînes de caractères.

Vérifications. La commande LaTeX **\newcommand** vérifie que le nom de la commande qu'on définit n'est pas déjà utilisé, contrairement à la primitive TeX **\def** (ou sa variante **\gdef**), c'est pourquoi il est prudent de préférer la première à la seconde. Sa variante **\newcommand*** vérifie de plus que l'argument de la commande ne contient pas de saut de ligne. Là encore, c'est celle-ci qu'il faut privilégier la plupart du temps.

3.4 Dans le fichier `lua`

En guise d'introduction au langage de programmation `lua`, nous donnons la définition de la fonction `luapoliste` appelée par la commande `\poliste`.

```
function luapoliste(luaexcar,liste)
   local lispo= string . explode ( liste, ",+" ) ;
   tex.sprint("\\begin{tabular}{l|l|l|l}");
   tex.sprint("Nom de la fonte & romain & \\textit{italique} &
\\textbf{gras} \\\\");
   tex.sprint("\\hline");
   for i=1, #lispo do
```

10. Il peut y avoir des couples d'accolades `'{ ... }'` entre les deux.

```
      tex.sprint("{\\fontspec{" .. lispo[i] .. "}" .. lispo[i] .. "} &
 {\\fontspec{" .. lispo[i] ..."}" .. luaexcar .. "} & {\\fontspec{" ..
lispo[i] .. "}\\textit{" .. luaexcar .. "}} & {\\fontspec{" .. lispo[
i] .. "}\\textbf{" .. luaexcar .. "}}\\\\  ") ;
   end
   tex.sprint("\\end{tabular}");
end
```

Comme on le voit, la syntaxe est assez simple :

— chaque nouvelle fonction, boucle, ... commence par un mot-clef (`function`, `for .. = .., .. do`, ...), termine par `end` et donne lieu à une nouvelle indentation ;

— à l'intérieur de chaque fonction, boucle, ... les lignes terminent par un point-virgule `;` (mais pas les deux lignes extrêmes) ;

— une variable est le plus souvent introduite avec `local` en début de ligne, pour limiter sa portée à la fonction courante ;

— la fonction est introduite (en première ligne) avec des variables pour tenir le rôle de ses arguments.

Plus spécifiquement, dans le code présenté ici, on voit apparaître

- l'instruction `string . explode ( .. , ",+'')` transforme son premier argument (nommé `liste` dans le code, c'est la deuxième chaîne de caractères passée par la commande `\poliste`) en une liste `lua` ; la variable locale `lispo` est donc une liste au sens de `lua`. Pour opérer cette transformation, l'instruction utilise son deuxième argument `",+"` : elle découpe la chaîne de caractères en morceaux en se basant sur les virgules qui y sont présentes. Ainsi la chaîne de caractères

  "Latin Modern Roman, Noto Serif, TeX Gyre Pagella, TeX Gyre Termes, TeX Gyre Heros, TeX Gyre Bonum"

  sera découpée en 6 morceaux, donc la liste `lispo` aura 6 éléments, chacun étant le nom d'une fonte.
- Une fois constituée la liste `lispo`, on accède à son cardinal avec `#lispo` et à son i-ème élément avec `lispo[i]`.
- Les instructions `tex.sprint` contiennent du code LaTeX à écrire dans le code virtuel :
  — celui-ci est placé entre doubles-apostrophes `" ... "` ;
  — chaque antislash \ du code LaTeX y est doublé : \\ ;
  — à l'intérieur d'une instruction `tex.sprint`, le code LaTeX peut-être concaténé avec des valeurs de variables `lua`, avec la syntaxe :

```
tex.sprint(" code \\LaTeX{} " .. variable .. " code \\LaTeX{} ");
```

Profitons de cet exemple pour insister sur le *doublement des antislash* dans le code LaTeX contenu dans les instructions `tex.sprint` !

Ainsi la deuxième instruction `tex.sprint`, qui écrit la première ligne du tableau (son entête), se termine par \\\\ qui donnera dans le code LuaLaTeX virtuel la double antislash \\ indiquant la fin de la première ligne du tableau.

Dans la boucle qui suit, la première instruction `tex.sprint` écrit dans le code virtuel, lorsque le premier argument de la fonction `luapoliste` est `"ABC abc 012"` et le deuxième `"Latin Modern Roman"` :

```
{\fontspec{Latin Modern Roman}Latin Modern Roman} & {\fontspec{Latin
Modern Roman}ABC abc 012} & {\fontspec{Latin Modern Roman}\textit{ABC
 abc 012}} & {\fontspec{Latin Modern Roman}\textbf{ABC abc 012}}\\
```

ce qui, en ajoutant les lignes du code formant l'environnement `tabular` et l'entête du tableau, donne :

Nom de la fonte	romain	*italique*	**gras**
Latin Modern Roman	ABC abc 012	*ABC abc 012*	**ABC abc 012**

Lorsqu'il y a plusieurs fontes passées en deuxième argument de la commande `\poliste`, la boucle ci-dessus écrit dans le code virtuel une ligne comme ci-dessus pour chaque élément de la liste `lispo`, c'est-à-dire pour chaque fonte incluse en argument.

Un autre exemple. La fonction `luapoliste` ci-dessus est inspirée du premier exemple présenté dans [Chu10], qui montre comment faire dresser par `lua` un tableau de données à partir d'un fichier de données externe, et introduit des instructions `lua` supplémentaires :

- `local case ={};` pour introduire une variable locale de format liste, initialisée à la liste vide ;
- `io . input ( fichier )` ; pour lire le fichier de données externe ;
- `for ligne in io . lines () do` pour parcourir les *lignes* de ce fichier ;
- `case = string . explode ( ligne , " +" )` ; pour découper chaque ligne en liste en utilisant les *espaces* comme séparateurs [11] ;
- les tests conditionnels

```
if j == #case and i == #ligne then
       ...
elseif j == #case then
       ...
else
       ...
end
```

et

```
if sepligne ~= " " then
   ...
end
```

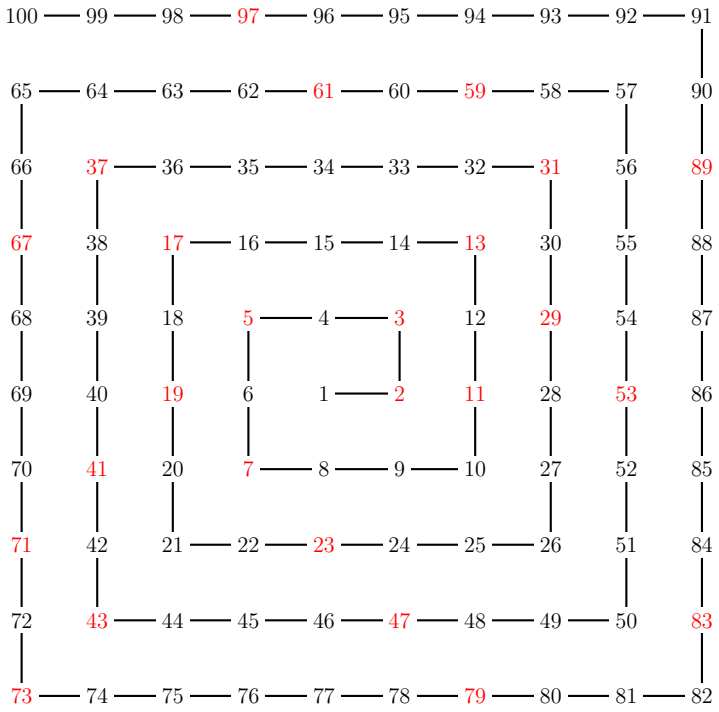
où == teste l'égalité et ~= teste la différence (ici avec la chaîne vide).

Noter ([Chu10, p. 39]) que l'auteur utilise `lua` pour mettre en forme les données (des lignes à 3 colonnes séparées par & et terminées par \\) mais qu'il introduit l'environnement `tabular` dans le code LaTeX, ce qui lui permet d'utiliser la même fonction `lua` pour présenter les mêmes données sous forme de matrice plutôt que de tableau.

11. L'auteur indique que pour utiliser les *passages à la ligne* comme séparateur, le deuxième argument de `string.explode` serait `"\n+"`. Voir aussi [Tea21, p. 61].

3.5 Avec des mathématiques et TikZ

La *spirale d'Ulam* est la représentation des entiers en spirale, démarrant à 1 au centre et tournant dans le sens inverse des aiguilles d'une montre, dans laquelle on repère les nombres premiers. Voici ce qu'on obtient en allant jusqu'à 100, avec les nombres premiers en rouge :



Stanislaw Ulam est un mathématicien qui, dans les années 1960, alors qu'il s'ennuyait en assistant à une conférence, a commencé à placer les entiers de cette façon sur une feuille, puis s'est mis à entourer les nombres premiers, essentiellement pour passer le temps. À sa grande surprise, ceux-ci ont semblé former des alignements dans les directions diagonales de son dessin. Il a ensuite, avec deux collaborateurs, confirmé cette impression en faisant tracer à l'un des plus puissants ordinateurs de l'époque des spirales beaucoup plus grandes (jusqu'à environ 65 000).

Se repérer dans la spirale. Sur le dessin ci-dessus, on remarque qu'on obtient un carré avec 1 au centre à chaque fois qu'on trace la spirale jusqu'au carré d'un entier impair : jusqu'à $9 = 3^2$, jusqu'à $25 = 5^2$, $49 = 7^2$, $81 = 9^2$. À chaque fois, le nombre final se trouve à l'extrêmité en bas à droite de la diagonale SE-NO du carré. Si l'on regarde de l'autre côté de la même diagonale, en haut à gauche, on voit que le nombre qui s'y trouve est toujours le carré d'un entier pair augmenté de 1 : $5 = 2^2+1$, $17 = 4^2+1$, $37 = 6^2+1$, $65 = 8^2+1$, $101 = 10^2+1$. Ces simples remarques permettent de se repérer dans la spirale : si m est un entier,

$$(2m+1)^2 = 4m^2+4m+1 \quad \text{et} \quad (2m)^2+1 = 4m^2+1$$

se trouvent aux extrêmités de la diagonale SE-NO du carré centré en 1 de côté $2m+1$ (on entend par là que $2m+1$ entiers se trouvent sur le côté de ce carré, entiers qu'on a mis dans une case dans la Figure 1).

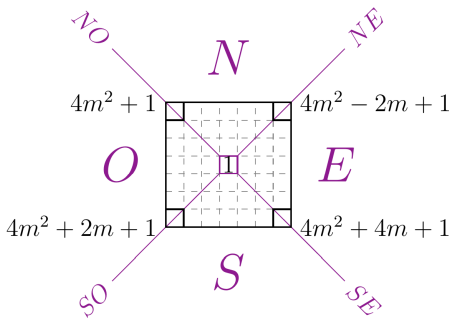


FIGURE 1 – La spirale d'Ulam jusqu'au carré de l'entier impair $2m+1$.

Pour repérer la position dans la spirale d'un entier n, en convenant que l'entier 1 est placé à la position $(0,0)$ et que les entiers sont à distance 1 de leur successeur (horizontalement ou verticalement), il suffit donc de :

(i) trouver le plus petit entier m tel que $n \leq (2m+1)^2$: cela signifie que n est sur le *bord* du carré de côté $2m+1$;

(ii) déterminer la différence $\delta = (2m+1)^2 - n$: comme $(2m-1)^2 < n \leq (2m+1)^2$, on a $0 \leq \delta < 8m$;

(iii) remarquer que $2m+1$ est à la position $(m,-m)$ donc :
- si $0 \leq \delta \leq 2m$, n est à la position $(m-\delta,-m)$;
- si $2m \leq \delta \leq 4m$, n est à la position $(-m,\delta-3m)$;
- si $4m \leq \delta \leq 6m$, n est à la position $(\delta-5m,m)$;
- si $6m \leq \delta < 8m$, n est à la position $(m,7m-\delta)$.

On a

$$n \leq (2m+1)^2 \iff m \geq \frac{\sqrt{n}-1}{2}$$

donc le plus petit entier m tel que $n \leq (2m+1)^2$ est $m = \left\lceil \frac{\sqrt{n}-1}{2} \right\rceil$. Il s'ensuit que la fonction `lua` suivante donne la position d'un entier `x` dans la spirale.

```
function position(x)
         local m=math.ceil((math.sqrt(x)-1)/2) ;
         local d=(2*m+1)^2-x ;
         local p={} ;
         local couleur=("black") ;
         if isPrime(x) then
            couleur=("red") ;
         end
         if d <= 2*m then
            p={m-d,-m,couleur} ;
         elseif d <= 4*m then
            p={-m,d-3*m,couleur} ;
         elseif d <= 6*m then
            p={d-5*m,m,couleur} ;
```

```
        else
           p={m,7*m-d,couleur} ;
        end
        return p
end
```

La position de l'entier x est contenue dans la variable locale de type liste p, intialisée à la liste vide et contenant 3 éléments : le premier pour l'abscisse, le deuxième pour l'ordonnée, le troisième est une chaîne de caractères, `"black"` ou `"red"`, selon la valeur en x de la fonction booléenne auxiliaire `isPrime`, qui renvoie `'True'` si x est premier, `'False'` sinon (fonction récupérée sur internet, des variantes existent sans doute dans certaines librairies de `lua`).

La fonction mathématique `math.sqrt` renvoie la racine carré de son argument, tandis que `math.ceil` renvoie la *partie entière supérieure* ('ceil' signifie 'plafond'), qu'on écrit mathématiquement $\lceil .. \rceil$: c'est le plus petit entier supérieur ou égal à l'argument de la fonction.

Enfin, la dernière instruction `return p` indique que la 'valeur' de la fonction `position` est donnée par la celle de la variable p à la fin du programme.

Dessiner la spirale. Ou plutôt la faire dessiner par `lua` !

Comme dans le premier exemple, on place le code `lua` dans un fichier à part (nommé cette fois `ulam.lua`). Celui-ci contient la fonction `position` qu'on vient de décrire, la fonction `isPrime` qu'on a copiée et une fonction `luaspirale` pour tracer la spirale jusqu'à un entier donné, qu'on décrit ci-dessous. On définit une commande LaTeX qui appelle cette fonction :

```
\luadirect { dofile( 'ulam.lua' ) }
\newcommand \spirale[1]{ \luadirect {luaspirale(#1)} }
```

Ici l'argument que la commande `\spirale` passe à la fonction `luaspirale` est simplement un entier, il n'est donc pas nécessaire d'utiliser la commande `\luastring` (ni d'encadrer `#1` par des double-apostrophes). Une fois la fonction `luaspirale` disponible, il ne restera qu'à inclure la commande `\spirale{100}` dans le code source pour obtenir le tracé du début de cette partie.

Voici le code de la fonction `luaspirale` :

```
  function luaspirale(n)
   local luascale=.7 ;
   tex.sprint("\\begin{tikzpicture}[scale=1]");
   tex.sprint("\\draw[thick] (0,0) node[fill=white,scale=" .. luascale
 .. ",inner sep=3pt]{$1$}");
   for i=2, n do
      tex.sprint("-- (" .. position(i)[1] .. "," .. position(i)[2] ..
") node[" .. position(i)[3] .. ",scale=" .. luascale .. ",inner sep=3
pt,fill=white]{$" .. i .. "$}") ;
   end
   tex.sprint(";");
   tex.sprint("\\end{tikzpicture}");
end
```

On peut noter :
- la variable locale `luascale` qui permet de régler – de l'intérieur du code de la fonction `luaspirale` – la taille des entiers dans la spirale ;
- le style des nœuds contenant les entiers, avec en option : la réduction de taille donnée par `luascale`, un fond blanc (pour détacher chaque entier du trait le reliant à ses deux voisins) et la "marge" autour de l'entier délimitée par `inner sep=3pt` (pour que le trait reste conséquent même entre des entiers à trois chiffres) ;
- toujours dans le style des nœuds contenant les entiers, la couleur dont ceux-ci sont écrits est définie par le troisième élément de la liste `position`, c'est-à-dire la chaîne de caractères `"red"` ou `"black"` selon qu'il est premier ou non.

Ne pas oublier de faire écrire par la fonction `luaspirale` dans le code source virtuel un point-virgule à la fin de la commande `\draw` de Ti*k*Z !

Ouvrages de référence. On trouvera énormément plus d'informations sur LuaTEX et LuaLATEX d'une part, sur la programmation en `lua` d'autre part, dans les ouvrages [Tea21] et [Ier16], ce dernier étant la 4e édition d'un ouvrage dont la première est librement accessible en ligne.

Références

[And15] Jacques André. "Utilisation locale de `\fontspec` pour des caractères spéciaux". 2015. URL : http://jacques-andre.fr/fontex/casseau+fontspec.pdf.

[BA22] Patrick Bideault et Jacques André. "La fonte de ce numéro : Infini. Analyse des propriétés d'une fonte OpenType". In : *La Lettre GUTenberg* 45 (mai 2022), p. 42-80. URL : https://www.gutenberg-asso.fr/Lettre-GUTenberg-45-mai-2022.

[Chu10] Maxime Chupin. "LuaLATEX pour les non-sorciers, deux exemples". In : *Cahiers GUTenberg* 54-55 (2010), p. 37-56. URL : http://cahiers.gutenberg.eu.org/fitem?id=CG_2010___54-55_37_0.

[Ier16] Roberto Ierusalimschy. *Programming in Lua*. Fourth Edition. 2016. URL : https://www.lua.org/pil/.

[Pég10a] Manuel Pégourié-Gonnard. "Un guide pour LuaLATEX". In : *Cahiers GUTenberg* 54-55 (2010), p. 13-35. URL : http://cahiers.gutenberg.eu.org/fitem?id=CG_2010___54-55_13_0.

[Pég10b] Manuel Pégourié-Gonnard. "Un guide pour LuaLATEX". In : *Cahiers GUTenberg* 54-55 (2010), p. 13-35. URL : http://cahiers.gutenberg.eu.org/fitem?id=CG_2010___54-55_13_0.

[Pég12] Manuel Pégourié-Gonnard. *The `luacode` package*. Version v1.2a. 23 jan. 2012. URL : http://mirrors.ctan.org/macros/luatex/latex/luacode/luacode.pdf.

[Rob24] Will Robertson. *The `fontspec` package. Font selection for XeLATEX and LuaLATEX*. Version v2.9a. 13 fév. 2024. URL : http://wspr.io/fontspec/.

[Tea21] LuaTEX development Team. *LuaTEX Reference Manual*. Version 1.10. 23 juill. 2021. URL : www.luatex.org.