

CHAPITRE 9

Codes source

Dans ce chapitre, on va voir comment faire pour insérer du texte brut dans un document. On utilise notamment cela pour insérer des codes source, des sorties de programme, ou toute autre texte qui ne doit pas être interprété comme du code \LaTeX . On verra également comment présenter joliment des algorithmes.

9.1 Insertion brute

Pour insérer du texte brut dans un document, on peut soit utiliser la commande `\verb`, soit l'environnement `verbatim`. La commande `\verb` est particulière en ce sens qu'on peut également l'utiliser avec des délimiteurs autres que des accolades ouvrantes et fermantes.

On obtient le logo \LaTeX avec la commande `\LaTeX`

La commande `\verb` permet d'insérer du texte brut

Dans cet exemple, on a utilisé le caractère `=` pour délimiter le paramètre de la commande `\verb`.



Code

```
On obtient le logo \LaTeX{} avec la commande \verb=\LaTeX=
\begin{verbatim}
  La commande \verb permet d'insérer du texte brut
\end{verbatim}
```

Notez que la commande `\verb` ne peut pas être utilisée dans un paramètre d'une autre commande.

9.1.1 Raccourci d'insertion brute

Cela peut vite devenir ennuyeux d'écrire `\verb` à chaque fois. Grâce au package `shortvrb` et sa commande `\MakeShortVerb`, il est possible de définir un caractère de telle sorte que tout texte entouré par ce caractère sera en mode brut.



Code

```
\MakeShortVerb{$}

On obtient le logo \LaTeX{} avec la commande $\LaTeX$
```

9.1.2 Conservation des tabulations

L'environnement `verbatimab` du package `moreverb` permet d'insérer du texte brut, mais en préservant les tabulations. L'option de l'environnement permet de spécifier la largeur des tabulations, en nombre d'espaces.

```
if (x > 10)
  print "Examen réussi"
```



Code

moreverb


```
\begin{verbatimab}[3]
if (x > 10)
  print "Examen réussi"
\end{verbatimab}
```

9.1.3 Ajout des numéros de ligne

Enfin, on peut ajouter les numéros des lignes en utilisant l'environnement `listing` du package `moreverb`. L'option de l'environnement indique l'écart entre les numéros, et son paramètre la première ligne qu'il faut numéroter.

```
1  for (i = 0 to 10)
2    if (i mod 2 = 0)
3      print i
```

Ici, pour modifier la largeur des tabulations, il faut manuellement redéfinir la commande `\verbatimabsize`.


 **Code** moreverb

```
\renewcommand{\verbatimtabsize}{3}
\begin{listing}[1]{1}
for (i = 0 to 10)
  if (i mod 2 = 0)
    print i
\end{listing}
```

L'environnement `listingcont` permet d'insérer un texte brut avec numérotation, cette dernière continuant celle du dernier listing inséré.

```
4      else
5      print i + 1
```

Cet environnement est très pratique lorsque vous devez présenter un programme par morceaux, séparés par des commentaires explicatifs.

 **Code** moreverb

```
\begin{listingcont}
else
  print i + 1
\end{listingcont}
```


9.1.4 Insertion depuis un fichier

On peut directement inclure un texte brut depuis un fichier texte. Pour cela, on va utiliser la commande `\verbatiminput` du package `verbatim`. Cette commande prend le chemin du fichier en paramètre.

Voici le contenu du fichier `data.csv` :

```
A,B,Somme
12,23,35
21,43,64
```

Vous pouvez également utiliser la commande `\verbatimtabinput` du package `moreverb` pour insérer un texte brut en préservant les tabulations. Pour avoir les numéros de ligne, vous pouvez utiliser `\listinginput`.

 **Code** verbatim

```
Voici le contenu du fichier \texttt{data.csv} :
\verbatiminput{data.csv}
```


9.1.5 Présenter un exemple

On peut présenter des exemples de code \LaTeX avec le résultat produit en utilisant l'environnement `SideBySideExample` du package `fvr-b-ex`.

\LaTeX , c'est fun :-)

1 `\LaTeX{}`, c'est fun :-)

Il faut définir la place disponible pour le résultat et le code avec l'option `xrightmargin`. On peut ensuite ajouter une bordure et la numérotation avec les options `frame` et `numbers`.

 **Code** fvr-b-ex


```
\begin{SideBySideExample}
  [xrightmargin=0.5\linewidth,frame=single,numbers=left]
  \LaTeX{ }, c'est fun :-)
\end{SideBySideExample}
```

9.2 Codes source d'un programme

Pour insérer le code source d'un programme, on peut utiliser le package `listings`. Ce dernier est très riche et offre de nombreuses options dont la coloration syntaxique. On utilise l'environnement `lstlisting` pour insérer un code source.

```
for (int i = 0; i < 5; i++)
{
    System.out.println ("i : " + i);
}
```

Comme on va le voir tout au long de cette section, ce package est très riche. Nous n'allons néanmoins pas tout explorer en détail, nous vous renvoyons à la documentation pour en savoir plus.

 **Code** listings

```
\begin{lstlisting}[language=java]
for (int i = 0; i < 5; i++)
{
    System.out.println ("i : " + i);
}
\end{lstlisting}
```

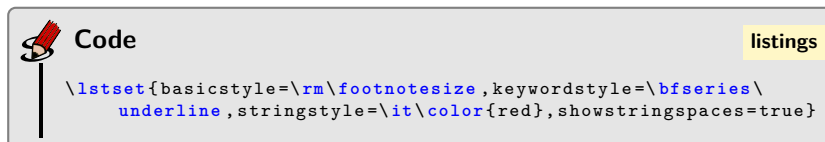
En plus de l'environnement `lstlisting`, on peut également utiliser la commande `\lstinline` qui fonctionne comme la commande `\verb`, en ce sens qu'on peut utiliser n'importe quel caractère comme délimiteur. Enfin, la commande `\lstinputlisting` permet de charger un code source depuis un fichier externe.

Il y a deux manières de préciser des options : soit via l'option de l'environnement ou de la commande, soit en utilisant la commande `\lstset` qui permet de définir des options de manière globale.

9.2.1 Style du texte

On peut modifier le style du texte en précisant la police, la couleur, la taille, etc. grâce aux options `basicstyle` (style de base), `keywordstyle` (mot réservé), `identifierstyle` (identificateur), `commentstyle` (commentaire) et `stringstyle` (chaîne de caractères). L'option `showstringspaces` permet de rendre les espaces dans les chaînes de caractères visibles.

```
for (int i = 0; i < 5; i++)
{
    System.out.println ( "i: " + i);
}
```



9.2.2 Numérotation

La numérotation est contrôlée par l'option `numbers` qu'on définit à `left`, `right` ou `none` (valeur par défaut). Le style des numéros est fixé par `numberstyle`, et le premier numéro est fixé avec `firstnumber` qui est soit un entier, soit `last` pour reprendre la numérotation. Enfin, `numbersep` représente l'écart entre les numéros affichés et `numberfirstline` indique s'il faut ou non numéroter la première ligne.

```
11  for (int i = 0; i < 5; i++)
12  {
    System.out.println ("i : " + i);
14 }
```

**Code****listings**

```
\lstset{numbers=left,numberstyle=\tiny\bfseries\underline,
stepnumber=2,firstnumber=11,numberfirstline=true}
```

9.2.3 Coloration syntaxique

On peut choisir le langage de programmation du code source présenté afin d'avoir la coloration syntaxique automatique. On utilise pour cela l'option `language`. Pour certains langages, il faut spécifier le dialecte en option. Par exemple, pour un programme en `C#`, on va écrire :

```
\lstset{language={ [Sharp]C }}
```

Notez qu'il faut englober le tout entre des accolades afin d'éviter des problèmes de compilation avec l'extension `keyval`.

9.2.4 Positionnement, dimensions et marge

On peut faire en sorte que le listing soit un flottant avec l'option `float`. On peut contrôler les marges avant et après le listing avec les options `aboveskip` et `belowskip`. Les options `xleftmargin` et `xrightmargin` spécifient les marges à gauche et à droite du listing. Enfin, l'option `linewidth` définit la largeur du listing.

On a ajouté une bordure dans l'exemple suivant, afin que vous puissiez voir les marges. On verra plus loin dans cette section comment faire cela.

```
for (int i = 0; i < 5; i++)
{
    System.out.println ("i : " + i);
}
```

**Code****listings**

```
\lstset{linewidth=9cm,xleftmargin=2cm,aboveskip=5mm,belowskip=1
cm,float=!h}
```


9.2.5 Cadre et couleur

On peut ajouter un cadre autour du listing avec l'option `frame`. On va utiliser les lettres `tblr` pour avoir une bordure en haut, en bas, à gauche et à droite. Les mêmes lettres en majuscule donneront une double bordure. Avec `rulesep`, on contrôle la distance entre les traits des bordures doubles. L'option `framesep` donne la distance entre le code et la bordure tandis que `framerule` donne l'épaisseur de cette dernière. On peut avoir des coins arrondis avec `frameround` qui prend quatre lettres (une par coin) en paramètre, ces lettres étant `t` ou `f` selon qu'on veuille ou non arrondir le coin.

On peut également jouer avec de la couleur. L'option `backgroundcolor` définit une couleur de fond. Les options `rulecolor`, `fillcolor` et `rulesepcolor` définissent la couleur de la bordure, du remplissage et de l'écart entre les doubles bordures.

```
for (int i = 0; i < 5; i++)
{
    System.out.println ("i : " + i);
}
```

Notez que certaines options ne sont pas compatibles. Il n'est par exemple pas possible d'utiliser `framerule` si `frameround` a été défini.



Code

```
\lstset{frame=tblr,rulesep=1mm,framesep=5mm,framerule=2pt,
xrightmargin=5mm,xleftmargin=5mm,rulecolor={\color[gray]
]{0.6}},rulesepcolor={\color[gray]{0.9}}}
```

listings


9.2.6 Légende, étiquette et liste des listings

On peut ajouter une légende avec l'option `caption` et une étiquette pour faire référence au listing avec `label`. Enfin, il est possible d'insérer la liste des listings d'un document avec la commande `\lstlistoflistings`.

```
for (int i = 0; i < 5; i++)
{
    System.out.println ("i : " + i);
}
```

Listing 9.1. Une boucle for en Java.

La légende est par défaut placée au dessus. On peut modifier cette position avec l'option `captionpos` qui vaut `b` (bas) ou `t` (haut).



Code

listings


```
\lstset{frame=lines,caption={Une boucle for en Java.},label=lst:
      java_for_loop,captionpos=b}
```

9.3 Algorithmes

Pour insérer des algorithmes, il est possible d'utiliser l'environnement `lstlisting` du package `listings` vu à la section précédente. Une solution plus pratique consiste à utiliser l'environnement `algorithm` du package `algorithm2e`.

```
x ← 5 ;
while x > 0 do
  | print x ;
  | x ← x + 1
end
```

Dans sa version basique, il suffit de placer le code de l'algorithme dans l'environnement `algorithm`. La commande `\gets` indique une affectation et la commande `\While` permet d'insérer une boucle. Chaque instruction simple doit se terminer par `\;`.



Code

algorithm2e

```
\begin{algorithm}
  $x \gets 5$ \;
  \While{$x > 0$}{
    print $x$ \;
    $x \gets x + 1$
  }
\end{algorithm}
```

9.3.1 Instructions prédéfinies

De nombreuses instructions sont prédéfinies. Voici les plus courantes :

- Entrées/Sorties : `\KwData`, `\KwResult`, `\KwIn` et `\KwOut`

- Intervalle : `\KwTo`
- Renvois de valeur : `\KwRet` ou `\Return`
- Conditions : `\If`, `\ElseIf`, `\Else` et `\eIf`
- Choix : `\Switch`, `\Case` et `\Other`
- Boucles : `\For`, `\While`, `\ForEach`, `\ForAll` et `\Repeat`

Algorithme 1: Un exemple d'algorithme.


Input : N un entier positif

```

 $sum \leftarrow 0$  ;
for  $x \leftarrow 0$  to  $N$  do
  if  $x$  est pair then
     $sum \leftarrow sum/2$  ;
  else
     $sum \leftarrow sum + 1$  ;
  end
end
return  $x$ 

```

Cet exemple utilise quelques-unes des instructions prédéfinies. De plus, on a déjà appliqué quelques modifications de style. On verra en détails ces modifications à la section 9.3.3. Remarquez également la commande `\BlankLine` qui permet d'insérer un petit espace vertical, ainsi que la commande `\caption` qui est utilisée pour définir la légende. Pour que le mot « *Algorithme* » apparaisse en français, il faut ajouter l'option `french` en important le package `algorithm2e`.



Code

algorithm2e

```

\begin{algorithm}
\caption{Un exemple d'algorithme.}
\KwIn{$N$ un entier positif}
\BlankLine
$sum \gets 0$ \;
\For{$x \gets 0$ \KwTo $N$}{
  \eIf{$x$ est pair}{
    $sum \gets sum / 2$ \;
  }{
    $sum \gets sum + 1$ \;
  }
}
\Return{$x$}
\end{algorithm}

```


9.3.2 Nouvelles instructions

On peut définir des nouvelles instructions avec différentes commandes. Les principales sont `\SetKw` et `\SetKwInput`.

Precondition : S un ensemble fini d'entiers

```
sum ← 0 ;
foreach  $i \in S$  do
|   sum ← sum + i ;
end
print sum ;
```

La commande `\SetKw` permet de définir un nouveau mot réservé et la commande `\SetKwInput` permet de définir une nouvelle instruction de type input.


Code

algorithm2e

```
\SetKw{KwPrint}{print}
\SetKwInput{KwPre}{Precondition}

\begin{algorithm}
\KwPre{ $SS$  un ensemble fini d'entiers}
\BlankLine
$sum \gets 0$ \;
\ForEach{$i$ \in $S$}{
    $sum \gets sum + i$ \;
}
\KwPrint{$sum$} \;
\end{algorithm}
```

9.3.3 Style

On peut modifier le style des algorithmes via les options du package lorsqu'on l'importe. Les options principales sont :

- `boxed` ou `boxruled` pour avoir un cadre autour de l'algorithme, avec la légende dedans ou pas ;
- `ruled` ou `algoruled` pour avoir des traits horizontaux avant et après l'algorithme, avec plus ou moins d'espace ;
- `lined`, `vlined` ou `noline` pour avoir des traits verticaux pour les instructions composées, ou rien du tout ;
- `linesnumbered` pour avoir les numéros de ligne ;

- `longend`, `shortend` ou `noend` pour des fins d'instructions longues ou courtes, ou aucune fin.

Enfin, on peut supprimer les points-virgules affichés par `\;` avec la commande `\DontPrintSemicolon`. On peut modifier la bordure avec la commande `\RestyleAlgo`. Les traits verticaux sont modifiables avec `\SetLine`, `\SetAlgoVlined` et `\SetNoline`. On peut obtenir les numéros de ligne avec `\LinesNumbered`.

Input : n un entier positif

Output : la valeur de la somme $1 + 2 + \dots + n$

```

1  $sum \leftarrow 0$ 
2 for  $i \leftarrow 1$  to  $n$  do
3    $sum \leftarrow sum + i$ 
4 return  $sum$ 

```



Code

algorithm2e

```

\RestyleAlgo{boxed}

\begin{algorithm}
\DontPrintSemicolon
\SetAlgoVlined
\LinesNumbered

\KwIn{$n$ un entier positif}
\KwOut{la valeur de la somme  $1 + 2 + \dots + n$ }
\BlankLine
$sum \gets 0$ \;
\For{$i \gets 1$ \KwTo $n$}{
  $sum \gets sum + i$ \;
}
\Return{$sum$} \;
\end{algorithm}

```

