

Diverses choses que je sais... sur la programmation avec L^AT_EX

version 1

Le T_EXnicien de surface

U.S.T.L. & GUTenberg

Dunkerque 2010 corrections et compléments 2010-06-17



Il s'agit de programmation modérément facile.

Il s'agit de programmation modérément facile.

Remerciements à Josselin NOIREL, Jean-Côme CHARPENTIER,
Manuel PÉGOURIÉ-GONNARD et à beaucoup d'autres intervenant
sur `fr.comp.text.tex`.

OUTILS CLASSIQUES, RÉVISION

Ce sont les commandes fournies par le noyau $\LaTeX 2_{\epsilon}$.

▶ `\newcommand` :

```
\newcommand\truc[3][a]{#3, #1 et #2};
```

OUTILS CLASSIQUES, RÉVISION

Ce sont les commandes fournies par le noyau $\LaTeX 2_{\epsilon}$.

- ▶ `\newcommand` :
- ▶ `\newcommand\truc[3][a]{#3, #1 et #2};`
- ▶ `\newcommand*`;

OUTILS CLASSIQUES, RÉVISION

Ce sont les commandes fournies par le noyau $\LaTeX 2_{\epsilon}$.

- ▶ `\newcommand` :
`\newcommand\truc[3][a]{#3, #1 et #2};`
- ▶ `\newcommand*`;
- ▶ `\renewcommand`;

OUTILS CLASSIQUES, RÉVISION

Ce sont les commandes fournies par le noyau L^AT_EX 2_ε.

- ▶ `\newcommand` :
`\newcommand\truc[3][a]{#3, #1 et #2};`
- ▶ `\newcommand*`;
- ▶ `\renewcommand`;
- ▶ `\renewcommand*`;

OUTILS CLASSIQUES, RÉVISION

Ce sont les commandes fournies par le noyau L^AT_EX 2_ε.

- ▶ `\newcommand` :
`\newcommand\truc[3][a]{#3, #1 et #2};`
- ▶ `\newcommand*`;
- ▶ `\renewcommand`;
- ▶ `\renewcommand*`;
- ▶ `\providecommand`;

OUTILS CLASSIQUES, RÉVISION

Ce sont les commandes fournies par le noyau L^AT_EX 2_ε.

- ▶ `\newcommand` :
`\newcommand\truc[3][a]{#3, #1 et #2};`
- ▶ `\newcommand*`;
- ▶ `\renewcommand`;
- ▶ `\renewcommand*`;
- ▶ `\providecommand`;
- ▶ `\DeclareRobustCommand`;

OUTILS CLASSIQUES, RÉVISION

Ce sont les commandes fournies par le noyau L^AT_EX 2_ε.

- ▶ `\newcommand` :
`\newcommand\truc[3][a]{#3, #1 et #2};`
- ▶ `\newcommand*`;
- ▶ `\renewcommand`;
- ▶ `\renewcommand*`;
- ▶ `\providecommand`;
- ▶ `\DeclareRobustCommand`;
- ▶ `\CheckCommand`;

OUTILS CLASSIQUES, RÉVISION

Ce sont les commandes fournies par le noyau L^AT_EX 2_ε.

- ▶ `\newcommand` :
`\newcommand\truc[3][a]{#3, #1 et #2};`
- ▶ `\newcommand*`;
- ▶ `\renewcommand`;
- ▶ `\renewcommand*`;
- ▶ `\providecommand`;
- ▶ `\DeclareRobustCommand`;
- ▶ `\CheckCommand`;
- ▶ `\newenvironment`;

OUTILS CLASSIQUES, RÉVISION

Ce sont les commandes fournies par le noyau L^AT_EX 2_ε.

- ▶ `\newcommand` :
- ▶ `\newcommand\truc[3][a]{#3, #1 et #2};`
- ▶ `\newcommand*`;
- ▶ `\renewcommand`;
- ▶ `\renewcommand*`;
- ▶ `\providecommand`;
- ▶ `\DeclareRobustCommand`;
- ▶ `\CheckCommand`;
- ▶ `\newenvironment`;
- ▶ `\renewenvironment`.

AVEC `xargs` DE MANUEL PÉGOURIÉ-GONNARD

`\newcommandx`

AVEC `xargs` DE MANUEL PÉGOURIÉ-GONNARD

`\newcommandx`, `\newcommandx*`

AVEC `xargs` DE MANUEL PÉGOURIÉ-GONNARD

`\newcommandx`, `\newcommandx*`, `\providecommandx`

AVEC `xargs` DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommandx, \newcommandx*, \providecommandx,  
\providecommandx*
```


AVEC `xargs` DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommandx, \newcommandx*, \providecommandx,  
\providecommandx*, \renewcommandx
```

AVEC `xargs` DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommandx, \newcommandx*, \providecommandx,  
\providecommandx*, \renewcommandx, \renewcommandx*
```

AVEC `xargs` DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommandx, \newcommandx*, \providecommandx,  
\providecommandx*, \renewcommandx, \renewcommandx*,  
\newenvironmentx
```

AVEC **xargs** DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommandx, \newcommandx*, \providecommandx,  
\providecommandx*, \renewcommandx, \renewcommandx*,  
\newenvironmentx, \renewenvironmentx
```

AVEC `xargs` DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommandx, \newcommandx*, \providecommandx,  
\providecommandx*, \renewcommandx, \renewcommandx*,  
\newenvironmentx, \renewenvironmentx, \CheckCommandx,
```

AVEC **xargs** DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommandx, \newcommandx*, \providecommandx,  
\providecommandx*, \renewcommandx, \renewcommandx*,  
\newenvironmentx, \renewenvironmentx, \CheckCommandx,  
\DeclareRobustCommandx.
```

AVEC **xargs** DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommandx, \newcommandx*, \providecommandx,  
\providecommandx*, \renewcommandx, \renewcommandx*,  
\newenvironmentx, \renewenvironmentx, \CheckCommandx,  
\DeclareRobustCommandx.
```

xargs tire parti de **xkeyval**

AVEC xargs DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommand, \newcommand*, \providecommand,  
\providecommand*, \renewcommand, \renewcommand*,  
\newenvironment, \renewenvironment, \CheckCommand,  
\DeclareRobustCommand.
```

xargs tire parti de xkeyval

```
\newcommand*\coord[3][1=1, 3=n]{%  
  (#2_{#1}, \ldots, #2_{#3})}
```


AVEC xargs DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommandx, \newcommandx*, \providecommandx,
\providecommandx*, \renewcommandx, \renewcommandx*,
\newenvironmentx, \renewenvironmentx, \CheckCommandx,
\DeclareRobustCommandx.
```

xargs tire parti de xkeyval

```
\newcommandx*\coord[3][1=1, 3=n]{%
  (#2_{#1}, \ldots, #2_{#3})}
```

Utilisation :

`\coord{x}` qui donne (x_1, \dots, x_n) ,

AVEC xargs DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommandx, \newcommandx*, \providecommandx,
\providecommandx*, \renewcommandx, \renewcommandx*,
\newenvironmentx, \renewenvironmentx, \CheckCommandx,
\DeclareRobustCommandx.
```

xargs tire parti de xkeyval

```
\newcommandx*\coord[3][1=1, 3=n]{%
  (#2_{#1}, \ldots, #2_{#3})}
```

Utilisation :

`\coord{x}` qui donne (x_1, \dots, x_n) ,

`\coord[3]{x}` qui donne (x_3, \dots, x_n) ,

AVEC **xargs** DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommandx, \newcommandx*, \providecommandx,
\providecommandx*, \renewcommandx, \renewcommandx*,
\newenvironmentx, \renewenvironmentx, \CheckCommandx,
\DeclareRobustCommandx.
```

xargs tire parti de **xkeyval**

```
\newcommandx*\coord[3][1=1, 3=n]{%
  (#2_{#1}, \ldots, #2_{#3})}
```

Utilisation :

`\coord{x}` qui donne (x_1, \dots, x_n) ,

`\coord[3]{x}` qui donne (x_3, \dots, x_n) ,

`\coord{x}[p]` qui donne (x_1, \dots, x_p) ,

AVEC xargs DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommandx, \newcommandx*, \providecommandx,
\providecommandx*, \renewcommandx, \renewcommandx*,
\newenvironmentx, \renewenvironmentx, \CheckCommandx,
\DeclareRobustCommandx.
```

xargs tire parti de xkeyval

```
\newcommandx*\coord[3][1=1, 3=n]{%
  (#2_{#1}, \ldots, #2_{#3})}
```

Utilisation :

`\coord{x}` qui donne (x_1, \dots, x_n) ,

`\coord[3]{x}` qui donne (x_3, \dots, x_n) ,

`\coord{x}[p]` qui donne (x_1, \dots, x_p) ,

`\coord[0]{x}[p]` qui donne (x_0, \dots, x_p) .

AVEC xargs DE MANUEL PÉGOURIÉ-GONNARD

```
\newcommandx, \newcommandx*, \providecommandx,
\providecommandx*, \renewcommandx, \renewcommandx*,
\newenvironmentx, \renewenvironmentx, \CheckCommandx,
\DeclareRobustCommandx.
```

xargs tire parti de xkeyval

```
\newcommandx*\coord[3][1=1, 3=n]{%
  (#2_{#1}, \ldots, #2_{#3})}
```

Utilisation :

`\coord{x}` qui donne (x_1, \dots, x_n) ,

`\coord[3]{x}` qui donne (x_3, \dots, x_n) ,

`\coord{x}[p]` qui donne (x_1, \dots, x_p) ,

`\coord[0]{x}[p]` qui donne (x_0, \dots, x_p) .

☞ Le nombre d'arguments est toujours limité à 9.

SUFFIXES AVEC `suffix` DE DAVID KASTRUP

Pour pouvoir écrire des macros comme `\R*` et `\R+`, on peut utiliser `suffix`.

SUFFIXES AVEC `suffix` DE DAVID KASTRUP

Pour pouvoir écrire des macros comme `\R*` et `\R+`, on peut utiliser `suffix`.

```
\newcommand\R[2][1=1,2=X]{%  
  \EnsembleDeNbr{R}{#1}{#2}{\ReculParent}}  
  
\WithSuffix\newcommand\R*{\R[*]}
```

SUFFIXES, SUITE

```
1 \def\TraiterLigne{\TraiterLigne@x}
2
3 \def\TraiterLigne@x#1{%
4   \traiterequation{#1}%
5   \gappto\LeSysteme{\endgraph}%
6   \arret}
7
8 \WithSuffix\def\TraiterLigne@x[#1]#2{%
9   \traiterequation{#2}%
10  \traiterindication{#1}%
11  \gappto\LeSysteme{\endgraph}%
12  \arret}
13
14 \WithSuffix\def\TraiterLigne(#1){%
15   \AjouterSaut{#1}%
16   \TraiterLigne@x}
```


SUFFIXES, SUITE

```

1  \def\TraiterLigne{\TraiterLigne@x}
2
3  \def\TraiterLigne@x#1{%
4    \traiterequation{#1}%
5    \gappto\LeSysteme{\endgraph}%
6    \arret}
7
8  \WithSuffix\def\TraiterLigne@x[#1]#2{%
9    \traiterequation{#2}%
10   \traiterindication{#1}%
11   \gappto\LeSysteme{\endgraph}%
12   \arret}
13
14  \WithSuffix\def\TraiterLigne(#1){%
15    \AjouterSaut{#1}%
16    \TraiterLigne@x}

```

On peut alors avoir `\TraiterLigne{...}`

SUFFIXES, SUITE

```

1 \def\TraiterLigne{\TraiterLigne@x}
2
3 \def\TraiterLigne@x#1{%
4   \traiterequation{#1}%
5   \gappto\LeSysteme{\endgraph}%
6   \arret}
7
8 \WithSuffix\def\TraiterLigne@x[#1]#2{%
9   \traiterequation{#2}%
10  \traiterindication{#1}%
11  \gappto\LeSysteme{\endgraph}%
12  \arret}
13
14 \WithSuffix\def\TraiterLigne(#1){%
15   \AjouterSaut{#1}%
16   \TraiterLigne@x}

```

On peut alors avoir `\TraiterLigne{...}`, `\TraiterLigne[...]{...}`

SUFFIXES, SUITE

```

1  \def\TraiterLigne{\TraiterLigne@x}
2
3  \def\TraiterLigne@x#1{%
4    \traiterequation{#1}%
5    \gappto\LeSysteme{\endgraph}%
6    \arret}
7
8  \WithSuffix\def\TraiterLigne@x[#1]#2{%
9    \traiterequation{#2}%
10   \traiterindication{#1}%
11   \gappto\LeSysteme{\endgraph}%
12   \arret}
13
14  \WithSuffix\def\TraiterLigne(#1){%
15    \AjouterSaut{#1}%
16    \TraiterLigne@x}

```

On peut alors avoir `\TraiterLigne{...}`, `\TraiterLigne[...]{...}`,
`\TraiterLigne(...){...}`

SUFFIXES, SUITE

```

1  \def\TraiterLigne{\TraiterLigne@x}
2
3  \def\TraiterLigne@x#1{%
4    \traiterequation{#1}%
5    \gappto\LeSysteme{\endgraph}%
6    \arret}
7
8  \WithSuffix\def\TraiterLigne@x[#1]#2{%
9    \traiterequation{#2}%
10   \traiterindication{#1}%
11   \gappto\LeSysteme{\endgraph}%
12   \arret}
13
14 \WithSuffix\def\TraiterLigne(#1){%
15   \AjouterSaut{#1}%
16   \TraiterLigne@x}

```

On peut alors avoir `\TraiterLigne{...}`, `\TraiterLigne[...]{...}`,
`\TraiterLigne(...){...}` ou encore `\TraiterLigne(...)[...]{...}`.

DU NOM DES MACROS

Il y a deux sortes de macros :

DU NOM DES MACROS

Il y a deux sortes de macros :

- ▶ celles du genre `\&` dont le nom est composé d'un seul caractère de catégorie autre que « lettres » ;

DU NOM DES MACROS

Il y a deux sortes de macros :

- ▶ celles du genre `\&` dont le nom est composé d'un seul caractère de catégorie autre que « lettres » ;
- ▶ celles du genre `\UneMacro` dont le nom est composé d'un nombre arbitraire de « lettres ».

DU NOM DES MACROS

Il y a deux sortes de macros :

- ▶ celles du genre `\&` dont le nom est composé d'un seul caractère de catégorie autre que « lettres » ;
- ▶ celles du genre `\UneMacro` dont le nom est composé d'un nombre arbitraire de « lettres ».

La catégorie — *catcode* — est un entier compris entre 0 et 15 que T_EX attribue à chaque lexème qu'il lit. Les « lettres » ont 11 pour *catcode*. Elles ne contiennent que les caractères alphabétiques de l'ASCII : a à z et A à Z.

DU NOM DES MACROS

Il y a deux sortes de macros :

- ▶ celles du genre `\&` dont le nom est composé d'un seul caractère de catégorie autre que « lettres » ;
- ▶ celles du genre `\UneMacro` dont le nom est composé d'un nombre arbitraire de « lettres ».

La catégorie — *catcode* — est un entier compris entre 0 et 15 que T_EX attribue à chaque lexème qu'il lit. Les « lettres » ont 11 pour *catcode*. Elles ne contiennent que les caractères alphabétiques de l'ASCII : a à z et A à Z.

Une fois attribué, le *catcode* est figé.

DU NOM DES MACROS

Il y a deux sortes de macros :

- ▶ celles du genre `\&` dont le nom est composé d'un seul caractère de catégorie autre que « lettres » ;
- ▶ celles du genre `\UneMacro` dont le nom est composé d'un nombre arbitraire de « lettres ».

La catégorie — *catcode* — est un entier compris entre 0 et 15 que T_EX attribue à chaque lexème qu'il lit. Les « lettres » ont 11 pour *catcode*. Elles ne contiennent que les caractères alphabétiques de l'ASCII : a à z et A à Z.

Une fois attribué, le *catcode* est figé.

Lors de la lecture du source, T_EX consomme les espaces qui suivent une macro du deuxième type et conserve le premier espace qui suit une macro du premier type comme dans `&_aa` : « & aa ».

Utilisation :

```
\csname Une Macro\endcsname
```

Utilisation :

```
\csname Une Macro\endcsname
```

Définition :

```
\expandafter\def\csname Une Macro\endcsname{%  
...}
```

Utilisation :

```
\csname Une Macro\endcsname
```

Définition :

```
\expandafter\def\csname Une Macro\endcsname{%  
...}
```

La macro `\expandafter` agit de la manière suivante : le source

`\expandafter L1 L2,`

où L_1 et L_2 sont deux lexèmes, est développé en

L_1 *(forme développée une fois de L_2)*.

Utilisation :

```
\csname Une Macro\endcsname
```

Définition :

```
\expandafter\def\csname Une Macro\endcsname{%  
...}
```

La macro `\expandafter` agit de la manière suivante : le source

`\expandafter L1L2`,

où L_1 et L_2 sont deux lexèmes, est développé en

L_1 (forme développée une fois de L_2).

Après un certain nombre de développement

`\expandafter L1\expandafter L2\expandafter L3L4`,

devient

$L_1L_2L_3$ (forme développée une fois de L_4).

Cependant que

`\expandafter \expandafter \expandafter L1L2`,

devient

L_1 (forme développée une fois du premier lexème du développement de L_2).

QUELQUES REMARQUES

T_EX n'interdit pas de redéfinir ses primitives

QUELQUES REMARQUES

T_EX n'interdit pas de redéfinir ses primitives à vos risques et périls, bien entendu.

Le code `\def\csname Une Macro\endcsname{truc}` redéfinit la macro `\csname` de telle sorte qu'il faut écrire `\csname Une Macro\endcsname` pour obtenir « truc ».

QUELQUES REMARQUES

T_EX n'interdit pas de redéfinir ses primitives à vos risques et périls, bien entendu.

Le code `\def\csname Une Macro\endcsname{truc}` redéfinit la macro `\csname` de telle sorte qu'il faut écrire

`\csname Une Macro\endcsname` pour obtenir « truc ».

Le code `\csname Une Macro\endcsname` semble donc produire ce que l'on veut mais il se pourrait que l'on se précipite vers quelques ennuis ! ; -)

QUELQUES REMARQUES

T_EX n'interdit pas de redéfinir ses primitives à vos risques et périls, bien entendu.

Le code `\def\csname Une Macro\endcsname{truc}` redéfinit la macro `\csname` de telle sorte qu'il faut écrire `\csname Une Macro\endcsname` pour obtenir « truc ».

Le code `\csname Une Macro\endcsname` semble donc produire ce que l'on veut mais il se pourrait que l'on se précipite vers quelques ennuis ! ; -)

Même si la macro `\AutreMacro` existe, le source `\csname AutreMacro\endcsname` amènera T_EX à signaler une erreur : « l'utilisation de `\csname` n'est pas en accord avec sa définition ».

```
\def\iflanguage#1{%  
  \expandafter\ifx\csname l@#1\endcsname\relax  
  \@nolanerr{#1}%  
  \else  
  \bbl@afterfi{\ifnum\csname l@#1\endcsname=\language  
    \expandafter\@firstoftwo  
    \else  
    \expandafter\@secondoftwo  
    \fi}%  
  \fi}
```

Utilisation :

```
\@nameuse{Une Macro}
```

Utilisation :

```
\@nameuse{Une Macro}
```

Bien entendu il faut que @ soit une lettre.

Utilisation :

```
\@nameuse{Une Macro}
```

Bien entendu il faut que @ soit une lettre.

Une lettre au sens de T_EX c.-à-d. de *catcode* 11. C'est le cas dans un fichier de style — c'est le boulot de `\usepackage` de s'en assurer — ou lorsque l'on fait précéder le code de `\makeatletter` et suivre de `\makeatother`.

Utilisation :

```
\@nameuse{Une Macro}
```

Bien entendu il faut que @ soit une lettre.

Une lettre au sens de T_EX c.-à-d. de *catcode* 11. C'est le cas dans un fichier de style — c'est le boulot de `\usepackage` de s'en assurer — ou lorsque l'on fait précéder le code de `\makeatletter` et suivre de `\makeatother`.

Définition :

```
\@namedef{Une Macro}{...}
```

Utilisation :

```
\@nameuse{Une Macro}
```

Bien entendu il faut que @ soit une lettre.

Une lettre au sens de T_EX c.-à-d. de *catcode* 11. C'est le cas dans un fichier de style — c'est le boulot de `\usepackage` de s'en assurer — ou lorsque l'on fait précéder le code de `\makeatletter` et suivre de `\makeatother`.

Définition :

```
\@namedef{Une Macro}{...}
```

En fait dans les sources de L^AT_EX, on lit :

```
\def\@namedef#1{\expandafter\def\csname #1\endcsname}
```


EXEMPLES D'UTILISATION 1

1° Dans le noyau L^AT_EX :

```
\def\pagestyle#1{%  
  \@ifundefined{ps@#1}%  
  \undefinedpagestyle  
  {\@nameuse{ps@#1}}}
```

EXEMPLES D'UTILISATION 1

1° Dans le noyau L^AT_EX :

```
\def\pagestyle#1{%  
  \@ifundefined{ps@#1}%  
  \undefinedpagestyle  
  {\@nameuse{ps@#1}}}
```

Lorsque L^AT_EX rencontre `\pagestyle` il cherche son argument. Si l'on a écrit — comme d'habitude — `\pagestyle{plain}` l'argument est `plain`. Il remplace alors `\pagestyle{plain}` par

```
\@ifundefined{ps@plain}\undefinedpagestyle{\@nameuse{ps@plain}}
```

EXEMPLES D'UTILISATION 1

1° Dans le noyau L^AT_EX :

```
\def\pagestyle#1{%  
  \@ifundefined{ps@#1}%  
  \undefinedpagestyle  
  {\@nameuse{ps@#1}}}
```

Lorsque L^AT_EX rencontre `\pagestyle` il cherche son argument. Si l'on a écrit — comme d'habitude — `\pagestyle{plain}` l'argument est `plain`. Il remplace alors `\pagestyle{plain}` par

```
\@ifundefined{ps@plain}\undefinedpagestyle{\@nameuse{ps@plain}}
```

Il développe maintenant `\@ifundefined` &c.

EXEMPLES D'UTILISATION 1

1° Dans le noyau L^AT_EX :

```
\def\pagestyle#1{%
  \@ifundefined{ps@#1}%
  \undefinedpagestyle
  {\@nameuse{ps@#1}}}
```

Lorsque L^AT_EX rencontre `\pagestyle` il cherche son argument. Si l'on a écrit — comme d'habitude — `\pagestyle{plain}` l'argument est `plain`. Il remplace alors `\pagestyle{plain}` par

```
\@ifundefined{ps@plain}\undefinedpagestyle{\@nameuse{ps@plain}}
```

Il développe maintenant `\@ifundefined` &c.

Le résultat est que si la macro `\ps@plain` n'est pas définie, le code de départ devient `\undefinedpagestyle` et que si elle est définie, le code se réduit à `\ps@plain`.

EXEMPLES D'UTILISATION 1

1° Dans le noyau L^AT_EX :

```
\def\pagestyle#1{%
  \@ifundefined{ps@#1}%
  \undefinedpagestyle
  {\@nameuse{ps@#1}}}
```

Lorsque L^AT_EX rencontre `\pagestyle` il cherche son argument. Si l'on a écrit — comme d'habitude — `\pagestyle{plain}` l'argument est `plain`. Il remplace alors `\pagestyle{plain}` par

```
\@ifundefined{ps@plain}\undefinedpagestyle{\@nameuse{ps@plain}}
```

Il développe maintenant `\@ifundefined` &c.

Le résultat est que si la macro `\ps@plain` n'est pas définie, le code de départ devient `\undefinedpagestyle` et que si elle est définie, le code se réduit à `\ps@plain`.

Et T_EX reprend le premier lexème — soit `\undefinedpagestyle` soit `\ps@plain` — et recommence à développer si c'est possible.

EXEMPLES D'UTILISATION 2

2^o Dans `xifthen` :

```
\@namedef{TE@cnttest@<}{%  
  \ifnum\@tempcnta<\@tempcntb AA\else AB\fi}
```

EXEMPLES D'UTILISATION 2

2^o Dans `xifthen` :

```
\@namedef{TE@cnttest@<}{%  
  \ifnum\@tempcnta<\@tempcntb AA\else AB\fi}
```

Ce code permet d'utiliser une macro dont le nom comporte le caractère < qui n'a certainement pas le *catcode* 11.

AVEC etoolbox

etoolbox fournit `\csdef`, `\csgdef`, `\csedef` et `\csxdef` en remplacement de `\@namedef`.

AVEC etoolbox

etoolbox fournit `\csdef`, `\csgdef`, `\csedef` et `\csxdef` en remplacement de `\@namedef`.

Il fournit `\csuse` pour `\@nameuse`.

```
\newcommand\A@age@x[3] [1=A@!***!,3=A@!***!]{%
  \ifstrequal{#1}{A@!***!}{%
    \csuse{A@age@x@A@tyenag}#2\A@|}{%
    \csuse{A@ageL@x@A@tyenag}#1\A@|#2\A@|}%
  \nottoggle{A@printage}%
  {\ifstrequal{#3}{A@!***!}%
    {}%
    {\csletcs{#3}{ageinyears}}}%
  {}}
```

AVEC etoolbox

etoolbox fournit `\csdef`, `\csgdef`, `\csedef` et `\csxdef` en remplacement de `\@namedef`.

Il fournit `\csuse` pour `\@nameuse`.

```
\newcommand\A@age@x[3] [1=A@!***!,3=A@!***!]{%
  \ifstrequal{#1}{A@!***!}{%
    \csuse{A@age@x@A@tyenag}#2\A@|}{%
    \csuse{A@ageL@x@A@tyenag}#1\A@|#2\A@|}%
  \nottoggle{A@primage}%
  {\ifstrequal{#3}{A@!***!}%
    {}%
    {\csletcs{#3}{ageinyears}}}%
  {}}
```

`\csletcs` et `\nottoggle` sont fournis par etoolbox.

AVEC etoolbox

etoolbox fournit `\csdef`, `\csgdef`, `\csedef` et `\csxdef` en remplacement de `\@namedef`.

Il fournit `\csuse` pour `\@nameuse`.

```
\newcommand\A@age@x[3][1=A@!***!,3=A@!***!]{%
  \ifstrequal{#1}{A@!***!}{%
    \csuse{A@age@x@A@tyenag}#2\A@|}{%
    \csuse{A@ageL@x@A@tyenag}#1\A@|#2\A@|}%
  \nottoggle{A@printage}%
  {\ifstrequal{#3}{A@!***!}%
    {}%
    {\csletcs{#3}{ageinyears}}}%
  {}}
```

`\csletcs` et `\nottoggle` sont fournis par etoolbox.

`\ifstrequal` est fourni par xstring.

```
\newcommand*{\csuse}[1]{%  
  \ifcsname#1\endcsname  
    \csname#1\expandafter\endcsname  
  \fi}
```

etoolbox SUITE

```
\newcommand*{\csuse}[1]{%  
  \ifcsname#1\endcsname  
    \csname#1\expandafter\endcsname  
  \fi}
```

```
\newrobustcmd*{\csdef}[1]{%  
  \expandafter\def\csname#1\endcsname}
```

`\newrobustcmd` est fourni par `etoolbox`.

Une macro \TeX définie avec `\def` n'admet pas qu'un de ses arguments, si elle en accepte, contienne un saut de paragraphe. En \LaTeX , c'est `\newcommand*` qui a cette propriété. Pour avoir le comportement contraire il faut utiliser `\newcommand` ou, en \TeX , `\long\def`.

etoolbox SUITE ENCORE

Une macro T_EX définie avec `\def` n'admet pas qu'un de ses arguments, si elle en accepte, contienne un saut de paragraphe. En L^AT_EX, c'est `\newcommand*` qui a cette propriété. Pour avoir le comportement contraire il faut utiliser `\newcommand` ou, en T_EX, `\long\def`.

Une macro T_EX définie avec `\def` dans un groupe n'est définie que dans ce groupe. Si l'on veut qu'elle soit définie également hors du groupe il faut employer `\global\def` ou `\gdef`.

Une macro T_EX définie avec `\def` n'admet pas qu'un de ses arguments, si elle en accepte, contienne un saut de paragraphe. En L^AT_EX, c'est `\newcommand*` qui a cette propriété. Pour avoir le comportement contraire il faut utiliser `\newcommand` ou, en T_EX, `\long\def`.

Une macro T_EX définie avec `\def` dans un groupe n'est définie que dans ce groupe. Si l'on veut qu'elle soit définie également hors du groupe il faut employer `\global\def` ou `\gdef`.

On peut obtenir que la définition de la macro soit développée lors de sa création avec `\edef`. `\xdef` en est la version `\global`.

Une macro T_EX définie avec `\def` n'admet pas qu'un de ses arguments, si elle en accepte, contienne un saut de paragraphe. En L^AT_EX, c'est `\newcommand*` qui a cette propriété. Pour avoir le comportement contraire il faut utiliser `\newcommand` ou, en T_EX, `\long\def`.

Une macro T_EX définie avec `\def` dans un groupe n'est définie que dans ce groupe. Si l'on veut qu'elle soit définie également hors du groupe il faut employer `\global\def` ou `\gdef`.

On peut obtenir que la définition de la macro soit développée lors de sa création avec `\edef`. `\xdef` en est la version `\global`.

On pourra rechercher ces commandes dans le fichier `source2e.pdf` pour des exemples d'utilisation.

- ▶ Sur T_EX

- ▶ Sur T_EX
 - ▶ *The T_EXbook*, D. E. Knuth ; traduction française *Le T_EXlivre*;

BIBLIOGRAPHIE

- ▶ Sur T_EX
 - ▶ *The T_EXbook*, D. E. Knuth ; traduction française *Le T_EXlivre* ;
 - ▶ *T_EX by topic*, V. Eijkhout, disponible chez <http://www.lulu.com/content/2555607> (version papier) et sur <https://savannah.nongnu.org/projects/texbytopic> (pdf).

BIBLIOGRAPHIE

- ▶ Sur T_EX
 - ▶ *The T_EXbook*, D. E. Knuth ; traduction française *Le T_EXlivre* ;
 - ▶ *T_EX by topic*, V. Eijkhout, disponible chez <http://www.lulu.com/content/2555607> (version papier) et sur <https://savannah.nongnu.org/projects/texbytopic> (pdf).
- ▶ Sur L^AT_EX

BIBLIOGRAPHIE

- ▶ Sur T_EX
 - ▶ *The T_EXbook*, D. E. Knuth ; traduction française *Le T_EXlivre* ;
 - ▶ *T_EX by topic*, V. Eijkhout, disponible chez <http://www.lulu.com/content/2555607> (version papier) et sur <https://savannah.nongnu.org/projects/texbytopic> (pdf).
- ▶ Sur L^AT_EX
 - ▶ Les sources commentées du noyau de L^AT_EX 2_ε sont disponibles sous le nom `source2e.pdf`.