Documentation du package pseudocode.sty
version 3.0

Léa Breban, Bénédicte Leterme, Mélodie Nonnon,
Nicolas Delestre

14 novembre 2023

Table des matières

Introduction

Ce package est destiné aux enseignants, chercheurs et étudiants de l'INSA de Rouen qui souhaitent afficher du pseudocode dans les documents LaTeX. La présentation et la syntaxe ont été créées de manière à ce que les algorithmes soient compréhensibles par tous. La mise en évidence des termes importants ainsi que l'indentation sont automatiques. Par ailleurs, tous les mots clés sont en français.

Utilisation du package

Il y a trois manières d'utiliser le package **pseudocode.sty** dans votre document.

Si vous n'avez pas les droits root

1. Ajoutez le package directement dans le même dossier que votre document. L'inconvénient de cette méthode est qu'il faut copier le package dans tous les dossiers où il est utilisé.
2. La méthode la plus propre consiste à installer une fois pour toutes le package sur la machine, peu importe l'emplacement de vos documents sources LaTeX. Il suffit de paramétrer la variable d'environnement **HOMETEXMF**, en lui donnant comme valeur l'emplacement où se trouve le package. Pour cela, ouvrez le fichier /.bashrc et ajoutez la commande suivante : `HOMETEXMF=chemin/vers/le/repertoire`. Mettez ensuite à jour la base de données LaTeX avec `texhash ~/repertoireParent`.

Si vous avez les droits root

Vous pouvez alors chercher le dossier **texmf** dans le système, puis de copier le package **pseudocode.sty** dans le dossier **tex** (Normalement, ce dossier se trouve dans `/usr/share/texmf/tex/`).

Implementation

Environnement

Toutes les commandes de ce package, à l'exception des TAD, doivent être utilisées au sein de l'environnement algorithme, donc entre les commandes `\begin{algorithme}` et `\end{alorithme}` Noter qu'un algorithme peut contenir plusieurs fonction(s) et procédure(s).

1 Déclarations

1.1 Types

Les types de base

Les commandes `\nomDuType` servent à utiliser les différents types existants. Voici les types disponibles :

Exemple :

Entier, **Naturel**, **NaturelNonNul**, **Reel**, **ReelPositif**, **ReelPositifNonNul**, **ReelNegatif**, **ReelNegatifNonNul**, **Booleen**, **Caractere**, **Chaine de caracteres**	`\entier, \naturel, \naturelNonNul, \reel, \reelPositif, \reelPositifNonNul, \reelNegatif, \reelNegatifNonNul, \booleen, \caractere, \chaine`

Tableau

La commande `\tableau` sert à déclarer un type tableau à n dimensions
Paramètres :
#1 : Intervalles des dimensions (indices du tableau), séparés par une virgule
#2 : de ou d'
#3 : Type des éléments du tableau

Exemple :

Tableau[1..MAX][1..MAX] **d' Entier**	`\tableau{1..MAX, 1..MAX}{d'}{\entier}`

Constante

La commande `\constante` sert à déclarer une constante
Paramètres :
#1 : Nom de la constante
#2 : Valeur de la constante

Exemple :

Constante MAX = 100	`\constante{MAX}{100}`

Enumeration

La commande `\typeEnumere` sert à déclarer un type énuméré
Paramètres :
#1 : Nom de l'énuméré
#2 : Enumération des valeurs possibles

Exemple :

Type Saisons = {ETE, AUTOMNE, HIVER, PRINTEMPS}

```
\begin{algorithme}
 \typeEnumere{Saisons}{ETE, AUTOMNE,
        HIVER, PRINTEMPS}
\end{algorithme}
```

Le type pointeur

La commande `\typePointeur` sert à déclarer un pointeur
Paramètres :
#1 : Type sur lequel le pointeur pointe

Exemple :

p : ^**Naturel**

```
p : \typePointeur{\naturel}
```

Le type structure

L'environnement `enregistrement` à utiliser entre `\begin{enregistrement}` et `\end{enregistrement}` sert à déclarer un type particulier : les structures.
Paramètres :
#1 : nom de la structure

Pour pouvoir l'alimenter, c'est-à-dire lui attribuer des champs, il faut utiliser la commande `\champEnregistrement`.
Paramètres :
#1 : nom du champ
#2 : type du champ

Pour accéder au champ d'une structure, on utilise la commande `\champ`.
Paramètres :
#1 : nom de la variable du type structure
#2 : nom du champ à accéder

Exemple :

Type Personne = **Structure**
    nom : **Chaine de caracteres**
    prenom : **Chaine de caracteres**
finstructure
fonction nomPersonne (p : Personne) **: Chaine de caracteres**
debut
    retourner p.nom
fin

```
 \begin{algorithme}
 \begin{enregistrement}{Personne}
    \champEnregistrement{nom}{\chaine}
    \champEnregistrement{prenom}{\chaine}
 \end{enregistrement}
\fonction{nomPersonne}{p : Personne}
 {\chaine}{}{}
 {\retourner{\champ{p}{nom}}}
\end{algorithme}
```

Définition d'un type

La commande `\type` permet de définir un nouveau type.
Paramètres :
#1 : Nom du nouveau type
#2 : Définition du type

Exemple :

Type Plateau = **Tableau**[1..MAX][1..MAX] **d' Entier**

```
\begin{algorithme}
  \type{Plateau}
   {\tableau{1..MAX, 1..MAX}
   {d'}{\entier}}
\end{algorithme}
```

1.2 Types Abstraits de Données TAD

TAD

La définition d'un tad se fait dans un environnement tad, entre `\begin{tad}` et `\end{tad}`.

`\tadNom` prend en paramètre le **nom** du TAD

`\tadParametres` prend en paramètre les différents **paramètres** du TAD

`\tadDependances` prend en paramètre les **dépendances** du TAD vis à vis d'autre types

Les définitions d'**opérations** doivent se situer dans l'environnement tadOperations, entre `\begin{tadOperations}{nomLePlusLong}` et `\end{tadOperations}`. Le nom le plus long parmi les opérations doit être mis en paramètre.
Les commandes `\tadOperation` et `\tadOperationAvecPrecondition` servent à définir les différentes opérations du TAD
Paramètres :
#1 : Nom de l'operation
#2 : Paramètres en entrée de l'opération (utiliser `\tadParams`)
#3 : Paramètres en sortie de l'opération(utiliser `\tadParams`)

Toutes les **semantiques**, définies grâce à la commande `\tadSemantique`, doivent être définies dans l'environnement tadSemantiques, entre `\begin{tadSemantiques}{nomLePlusLong}` et `\end{tadSemantiques}`. Le nom le plus long parmi les sémantiques doit être mis en paramètre.

Tous les **axiomes**, définis grâce à la commande `\tadAxiome`, doivent être utilisés dans l'environnement tadAxiomes, entre `\begin{tadAxiomes}` et `\end{tadAxiomes}`.

Toutes les **préconditions** d'operations doivent être definies dans un environnement tadPreconditions, entre `\begin{tadPreconditions}{nomLePlusLong}` et `\end{tadPreconditions}`. Le nom le plus long parmi les préconditions doit être mis en paramètre.
La commande `\tadPrecondition` sert à définir une précondition
Paramètres :
#1 : Nom de l'operation
#2 : la précondition

Exemple :

```
\begin{tad}
 \tadNom{Liste}
 \tadParametres{Element}
 \tadDependances{\booleen, \naturelNonNul, \naturel}
 \begin{tadOperations}{obtenirElement}
   \tadOperation{liste}{}{\tadParams{Liste}}
   \tadOperation{estVide}{\tadParams{Liste}}{\tadParams{Booleen}}
   \tadOperationAvecPreconditions{inserer}{\tadParams{Liste, \naturelNonNul, Element}}{\tadParams{Liste}}
   \tadOperationAvecPreconditions{supprimer}{\tadParams{Liste, \naturelNonNul}}
                                  {\tadParams{Element}}
   \tadOperationAvecPreconditions{obtenirElement}{\tadParams{Liste, \naturelNonNul}}{\tadParams{Element}}
   \tadOperation{longueur}{\tadParams{Liste}}{\tadParams{\naturel}}
 \end{tadOperations}
 \begin{tadSemantiques}{longueur}
   \tadSemantique{liste}{cree une liste vide}
   \tadSemantique{inserer}{insere un element a une position donnee}
   \tadSemantique{longueur}{renvoie la longueur de la liste}
 \end{tadSemantiques}
 \begin{tadAxiomes}
   \tadAxiome{estVide(liste())}
   \tadAxiome{non estVide(inserer(l,i,e))}
   \tadAxiome{supprimer(inserer(l,i,e),i)=l}
   \tadAxiome{obtenirElement(inserer(inserer(l,i,e1),i,e2),i+1)=e1}
   \tadAxiome{longueur(liste())=0}
   \tadAxiome{longueur(inserer(l,i,e))=1+longueur(l)}
 \end{tadAxiomes}
 \begin{tadPreconditions}{obtenirElement(l,i)}
   \tadPrecondition{inserer(l,i,e)}{i $\leq$ longueur(l)+1}
   \tadPrecondition{supprimer(l,i)}{i $\leq$ longueur(l)}
   \tadPrecondition{obtenirElement(l,i)}{i $\leq$ longueur(l)}
 \end{tadPreconditions}
\end{tad}
```

Nom: Liste

Paramètre: Element

Utilise: **Booleen**, **NaturelNonNul**, **Naturel**

Opérations:

liste:		$\rightarrow$ Liste
estVide:	Liste	$\rightarrow$ Booleen
inserer:	Liste $\times$ **NaturelNonNul** $\times$ Element	$\nrightarrow$ Liste
supprimer:	Liste $\times$ **NaturelNonNul**	$\nrightarrow$ Element
obtenirElement:	Liste $\times$ **NaturelNonNul**	$\nrightarrow$ Element
longueur:	Liste	$\rightarrow$ **Naturel**

Sémantiques:

liste:	cree une liste vide
inserer:	insere un element a une position donnee
longueur:	renvoie la longueur de la liste

Axiomes:

- $estVide(liste())$
- $nonestVide(inserer(l, i, e))$
- $supprimer(inserer(l, i, e), i) = l$
- $obtenirElement(inserer(inserer(l, i, e1), i, e2), i + 1) = e1$
- $longueur(liste()) = 0$
- $longueur(inserer(l, i, e)) = 1 + longueur(l)$

Préconditions:

inserer(l,i,e):	i $\leq$ longueur(l)+1
supprimer(l,i):	i $\leq$ longueur(l)
obtenirElement(l,i):	i $\leq$ longueur(l)

2 Algorithme

2.1 Les Fonctions et Procédures

Type Fonction

La commande `\typeFonction` est à utiliser comme deuxième paramètre de la commande `\type`
Paramètres :
#1 : les paramètres
#2 : le type de retour
#3 : les préconditions (peut être vide)

Exemple :

Type ComparerDeuxEntiers =
fonction(a,b : **Entier**) : **Booleen**

```
\begin{algorithme}
   \type{ComparerDeuxEntiers}{
     \typeFonction{a,b : \entier}
     {\booleen}{}}
\end{algorithme}
```

Signature Fonction

La commande `\signatureFonction` est utilisée pour définir la signature d'une fonction
Paramètres :
#1 : Nom de la fonction
#2 : Paramètres formels
#3 : Type de retour
#4 : Préconditions (peut être vide)

Exemple :

fonction division (a, b : **Reel**) : **Reel**
⌊**précondition(s)** b ≥ 0

```
\begin{algorithme}
\signatureFonction{division}
   {a, b : \reel}{\reel}
   {b $\geq$ 0}
\end{algorithme}
```

Fonction

La commande `\fonction` sert à définir une fonction
Paramètres :
#1 : Nom de la fonction
#2 : Paramètres formels
#3 : Type de retour
#4 : Préconditions (peut être vide)
#5 : Déclaration des variables locales (peut être vide)
#6 : Instructions

Exemple :

fonction division (a, b : **Reel**) : **Reel**
⌊**précondition(s)** b ≥ 0
debut
retourner a/b
fin

```
\begin{algorithme}
\fonction{division}
   {a, b : \reel}{\reel}
   {b $\geq$ 0}{}
{\retourner{a/b}}
\end{algorithme}
```

Paramètres des Procédures

Les commandes `\paramEntree`, `\paramSortie`, et `\paramEntreeSortie` permettent de définir le type de passage de paramètre des paramètres formels d'une procédure.
(Les deux backslash `\\` *permettent juste de sauter une ligne pour des raisons esthétiques mais ils ne sont pas utiles)*

Exemple :

E r : **Reel**
E/S tab : **Tableau**[1..MAX] **deReel**
S ajoutReussi : **Booleen**

```
\paramEntree{r : \reel} \\
\paramEntreeSortie{tab :
    \tableauUneDimension{1..MAX}
      {de}{\reel}} \\
\paramSortie{ajoutReussi : \booleen} \\
```

Type procédure

La commande `\typeProcedure` est utilisée comme deuxième paramètre de la commande `\type`
Paramètres :
#1 : les paramètres formels
#2 : les préconditions (peut être vide)

Exemple :

Type AfficherEtatDuJeu =
procédure(**E** nbObjets : **Naturel**)

```
\begin{algorithme}
\type{AfficherEtatDuJeu}{
   \typeProcedure{
     \paramEntree{nbObjets : \naturel}
   }{}
}
\end{algorithme}
```

Signature Procédure

La commande `\signatureProcedure` est utilisée pour définir la signature d'une procédure
Paramètres :
#1 : Nom de la procedure
#2 : Paramètres formels
#3 : Préconditions (peut être vide)

Exemple :

procédure ajoutValeur **(E** r : **Reel E/S** tab : **Tableau**[1..MAX] **deReel S** ajoutReussi : **Booleen)**

```
\begin{algorithme}
\signatureProcedure{ajoutValeur}
   {\paramEntree{r : \reel}
     \paramEntreeSortie{tab :
        \tableauUneDimension{1..MAX}
         {de}{\reel}}
     \paramSortie{ajoutReussi : \booleen}}
   {}
\end{algorithme}
```

Procédure

La commande `\procedure` sert à écrire une procédure en pseudocode.
Paramètres :
#1 : Nom de la procedure
#2 : Paramètres formels
#3 : Préconditions (peut être vide)
#4 : Déclaration des variables locales (peut être vide)
#5 : Corps de la procédure

Exemple :

procédure ajoutValeur **(E** r : **Reel E/S** tab : **Tableau**[1..MAX] **de Reel S** ajoutReussi : **Booleen)**
debut
    ajoutReussi ← faux
    si taille ≤ MAX **alors**
        taille ← taille+1
        tab[taille] ← r
        ajoutReussi ← vrai
    finsi
fin

```
\begin{algorithme}
\procedure{ajoutValeur}{
     \paramEntree{r : \reel}
     \paramEntreeSortie{tab :
        \tableau{1..MAX}
          {de}{\reel}}
     \paramSortie{ajoutReussi : \booleen}}
{}{}
{  \affecter{ajoutReussi}{faux}
   \sialors{taille $\leq$ MAX}{
       \affecter{taille}{taille+1}
       \affecter{tab[taille]}{r}
       \affecter{ajoutReussi}{vrai}}}
\end{algorithme}
```

2.2 Instructions des algorithmes

Affecter

La commande `\affecter` sert à donner une valeur à une variable.
Paramètres :
#1 : la variable
#2 : la valeur qu'on veut lui donner

Exemple :

procédure echanger **(E/S** a, b : **Reel)**
    Déclaration temp : **Reel**
debut
    temp ← a
    a ← b
    b ← temp
fin

```
\begin{algorithme}
   \procedure{echanger}
     {\paramEntreeSortie{a, b : \reel}}
          {}{temp : \reel}
    {\affecter{temp}{a}
        \affecter{a}{b}
        \affecter{b}{temp}}
\end{algorithme}
```

Retourner

La commande `\retourner` sert à définir ce qu'une fonction va renvoyer.
Paramètres :
#1 : ce qu'on veut renvoyer

Exemple :

fonction multiplier (a, b : **Reel**) **: Reel**
debut
    retourner a*b
fin

```
\begin{algorithme}
   \fonction{multiplier}{a, b : \reel}
    {\reel}{}{}
    {\retourner{a*b}}
\end{algorithme}
```

Lire

La commande `\lire` sert à lire ce que l'utilisateur rentre par l'entrée standard (au clavier).
Paramètres :
#1 : la variable à laquelle on veut affecter une valeur

Exemple :

fonction calculerPuissance2 **() : Reel**
    Déclaration nbUser : **Reel**
debut
    lire(nbUser)
    retourner nbUser*nbUser
fin

```
\begin{algorithme}
   \fonction{calculerPuissance2}{}{\reel}{}
    {nbUser : \reel}
    {\lire{nbUser}
        \retourner{nbUser*nbUser}}
\end{algorithme}
```

Ecrire

La commande `\ecrire` sert à afficher du contenu dans le périphérique de sortie standard (l'écran la plupart du temps).
Paramètres :
#1 : ce qu'on veut afficher : soit directement une chaîne de caractères, soit une variable et dans ce cas son contenu s'affichera

Exemple :

procédure afficherBestNb **()**
    Déclaration bestNb : **Reel**
debut
    bestNb ← 42
    ecrire("Le meilleur nombre est : ")
    ecrire(bestNb)
fin

```
\begin{algorithme}
   \procedure{afficherBestNb}{}{}
    {bestNb : \reel}
        {\affecter{bestNb}{42}
        \ecrire{"Le meilleur nombre est : "}
        \ecrire{bestNb}}
\end{algorithme}
```

Instruction

La commande `\instruction` est à utiliser pour toute autre instruction non présente parmi celles présentées ci-dessus.
Paramètres :
#1 : nom de l'instruction

Exemple :

procédure afficher **()**
debut
    afficherBestNb()
fin

```
\begin{algorithme}
   \procedure{afficher}{}{}
    {}
        {\instruction{afficherBestNb()}}
\end{algorithme}
```

2.3 Les Conditionnelles

Si ... alors ... (sinon)

La commande `\sialorssinon` sert à réaliser une boucle conditionnelle si sinon, c'est à dire réaliser une instruction si la condition est vérifiée et une autre sinon
Paramètres :
#1 : Condition
#2 : Instruction à réaliser si la condition est vraie
#3 : Instruction à réaliser si la condition est fausse (peut être vide)

Exemple :

si a $\leq$ b **alors**
    retourner a
sinon
    retourner b
finsi

```
\begin{algorithme}
   \sialorssinon{a $\leq$ b}
   {\retourner{a}}
   {\retourner{b}}
\end{algorithme}
```

Cas où .. vaut

La commande `\cas` sert à réaliser un switch case.
Paramètres :
#1 : La variable pour laquelle on fait le switch case
#2 : Les différents cas (`\casclausegenerale` ou `\casclauseautre`)

La commande `\casclausegenerale` sert à définir une clause générale dans un switch case.
Paramètres :
#1 : La valeur de la variable
#2 : L'instruction à effectuer si la variable vaut cette valeur

La commande `\casclauseautre` sert à définir la clause par défaut dans un switch case.
Paramètre :
#1 : L'instruction par défaut à effectuer si toutes les autres clauses sont fausses

Exemple :

cas où a **vaut**
    1:
        ecrire("a=1")
    2:
        ecrire("a=2")
    autre :
        ecrire("a=0")
fincas

```
\begin{algorithme}
   \cas{a}{
   {\casclausegenerale{1}{\ecrire{"a=1"}}}
   {\casclausegenerale{2}{\ecrire{"a=2"}}}
   {\casclauseautre{\ecrire{"a=0"}}}}
\end{algorithme}
```

2.4 Les Itérations

2.4.1 Les itérations déterministes

Pour

La commande `\pour` sert à réaliser une itération déterministe `pour`
Paramètres :
#1 : Variable
#2 : Borne inférieure
#3 : Borne supérieure
#4 : Le pas (peut être vide et dans ce cas pas=1)
#5 : Instruction(s)

Exemple :

pour i ←1 **à** 10 **faire**
  ecrire(i)
finpour

```
\begin{algorithme}
\pour{i}{1}{10}{}{
   \ecrire{i}}
\end{algorithme}
```

Pour chaque

La commande `\pourChaque` sert à réaliser une itération déterministe `pour chaque`
Paramètres :
#1 : Variable
#2 : Liste
#3 : Instruction(s)

Exemple :

Type Saisons = {ETE, AUTOMNE, PRINTEMPS, HIVER}
pour chaque s **de** Saisons
  ecrire(s)
finpour

```
\begin{algorithme}
\typeEnumere{Saisons}
 {ETE, AUTOMNE, PRINTEMPS, HIVER}
\pourChaque{s}{Saisons}{
   \ecrire{s}}
\end{algorithme}
```

2.4.2 Les itérations indéterministes

Tant Que

La commande `\tantque` sert à réaliser une boucle indéterministe `tant que`.
Paramètres :
#1 : Condition
#2 : Instruction(s) à réaliser tant que la condition est vraie

Exemple :

i ← 0
tant que i ≤ 10 **faire**
  ecrire(i)
  i ← i+1
fintantque

```
\begin{algorithme}
   \affecter{i}{0}
   \tantque{i $\leq$ 10}{
     \ecrire{i}
     \affecter{i}{i+1}
   }
\end{algorithme}
```

Répéter ... jusqu'à ce que

La commande `\repeter` sert à réaliser une boucle indéterministe `répéter...jusqu'à ce que`.
Paramètres :
#1 : Condition
#2 : Instruction(s) à réaliser jusqu'à ce que la condition soit fausse

Exemple :

i ← 0
repeter
    ecrire(i)
    i ← i+1
jusqu'a ce que i ≥ 10

```
\begin{algorithme}
   \affecter{i}{0}
   \repeter{
     \ecrire{i}
     \affecter{i}{i+1}
   }{i $\geq$ 10}
\end{algorithme}
```

2.5 Les Pointeurs

Les pointeurs

Les commandes, `\pointeur`, `\pointeurNULL`, `\allouer`, `\desallouer`, `\adresse` servent à utiliser les pointeurs dans un algorithme.

Exemple :

procédure unPointeur (i : **Naturel**)
    Déclaration p : ^**Naturel**
debut
    p^ ← **null**
    allouer(p)
    p^ ← i
    p ← @i
    ecrire(p^)
    desallouer(p)
fin

```
\begin{algorithme}
 \procedure{unPointeur}{i : \naturel}{}
   {p : \typePointeur{\naturel}}
   {\affecter{\pointeur{p}}{\pointeurNULL}
   \instruction{\allouer{p}}
   \affecter{\pointeur{p}}{i}
   \affecter{p}{\adresse{i}}
   \ecrire{\pointeur{p}}
   \instruction{\desallouer{p}}}
\end{algorithme}
```

Champs pointeurs

La commande `\champPointeur` sert à accéder au champ d'un pointeur sur une structure.

Exemple :

Type Date = **Structure**
    jour : **Naturel**
    mois : **Naturel**
    annee : **Naturel**
finstructure
fonction obtenirJour **() : Naturel**
    Déclaration p : ^Date
debut
    retourner p→jour
fin

```
\begin{algorithme}
 \begin{enregistrement}{Date}
   \champEnregistrement{jour}{\naturel}
   \champEnregistrement{mois}{\naturel}
   \champEnregistrement{annee}{\naturel}
 \end{enregistrement}
 \fonction{obtenirJour}{}{\naturel}{}
   {p : \typePointeur{Date}}
   {\retourner{\champPointeur
                  {p}{jour}}}
\end{algorithme}
```

2.6 Commenter son code

Remarque et Commentaire

Il est important de rappeler que pour la maintenabilité d'un code, ainsi que sa lisibilité, il faut le documenter. C'est pour cela qu'il existe ces deux commandes.

Exemple :

fonction egalite (a,b : **Entier) : Booleen**
debut
  retourner a=b *// retourne VRAI si a=b*
fin
Remarque : *Cette fonction est peu utile.*

```
\begin{algorithme}
 \fonction{egalite}{a,b : \entier}
   {\booleen}{}{}
   {\retourner{a=b}
     \commentaire{retourne VRAI si a=b}}
  \remarque{Cette fonction est peu utile.}
\end{algorithme}
```

Paramétrage des mots-clefs

Renommer un mot clef

Vous pouvez modifier tous les mots clefs selon ce qui vous convient le mieux grâce à la commande :
`\renewcommand{\ancienNomMotClef}[0]{nouveauNom}`

Exemple :

if a $\leq$ b **then**
    retourner a
else
    retourner b
end

```
\renewcommand{\motclefSi}{if}
\renewcommand{\motclefAlors}{then}
\renewcommand{\motclefSinon}{else}
\renewcommand{\motclefFinsi}{end}
\begin{algorithme}
   \sialorssinon{a $\leq$ b}
   {\retourner{a}}
   {\retourner{b}}
\end{algorithme}
```

Voici la liste de tous les mots clefs qui existent :

La commande	Le mot clef
`\motclefAffectation`	$\leftarrow$
`\motclefAlgoDebut`	debut
`\motclefAlgoFin`	fin
`\motclefEntier`	Entier
`\motclefNaturel`	Naturel
`\motclefNaturelNonNul`	NaturelNonNul
`\motclefReel`	Reel
`\motclefReelPositif`	ReelPositif
`\motclefReelPositifNonNul`	ReelPositifNonNul
`\motclefReelNegatifNonNul`	ReelNegatifNonNul
`\motclefBooleen`	Booleen
`\motclefCaractere`	Caractere
`\motclefChaine`	Chaine de caracteres
`\motclefRemarque`	Remarque
`\motclefCommentaire`	//
`\motclefType`	Type
`\motclefDeclaration`	Déclaration
`\motclefConstante`	Constante
`\motclefTableau`	Tableau
`\motclefStructure`	Structure
`\motclefFinstructure`	finstructure
`\motclefSi`	si
`\motclefAlors`	alors
`\motclefSinon`	sinon
`\motclefFinsi`	finsi
`\motclefCasou`	cas où
`\motclefCasouVaut`	vaut
`\motclefFincas`	fincas
`\motclefTantque`	tant que
`\motclefTantqueFaire`	faire
`\motclefFintantque`	fintantque
`\motclefRepeter`	repeter
`\motclefJusquaceque`	jusqu'a ce que
`\motclefPour`	pour

\motclefPourA	à
\motclefPourPas	pas de
\motclefPourFaire	faire
\motclefFinPour	finpour
\motclefPourChaque	pour chaque
\motclefPourChaqueDe	de
\motclefEcrire	ecrire
\motclefLire	lire
\motclefPreconditions	précondition(s)
\motclefFonction	fonction
\motclefRetourner	retourner
\motclefProcedure	procédure
\motclefParamEntree	E
\motclefParamSortie	S
\motclefParamEntreeSortie	E/S
\motclefTADNom	Nom
\motclefTADParametre	Paramètre
\motclefTADUtilise	Utilise
\motclefTADOperations	Opérations
\motclefTADSemantiques	Sémantiques
\motclefTADAxiomes	Axiomes
\motclefTADPreconditions	Préconditions
\motclefPointeurNULL	null
\motclefAllouer	allouer
\motclefDesallouer	desallouer
\motclefDereferencer	^
\motclefReferencer	@