Package algorithme.sty

Version 2.3.1

Nicolas Delestre

13 novembre 2003

1 Introduction

L'objectif de ce package est de pouvoir écrire proprement des documents contenant des algorithmes au sens large. Il a été crée pour la rédaction du cours d'informatique que nous donnons en premier cycle INSA (Cf. `http://asi.insa-rouen.fr/enseignement/siteUV/algo/PremierCycle/`)

Avec ce package, vous allez pouvoir :

1. écrire des algorithmes :

   Nom: somme
   Role: Calculer la somme des n premiers entiers positifs, s=0+1+2+...+n
   Entrée: n : **Naturel**
   Sortie: s : **Naturel**
   Déclaration: i : **Naturel**
   début
   s ← 0
   pour i ←0 **à** n **faire**
   s ← s+i
   finpour
   fin

2. écrire des fonctions :

   fonction abs **(**unEntier : **Entier) : Entier**
   début
   si unEntier ≥ 0 **alors**
   retourner unEntier
   sinon
   retourner -unEntier
   finsi
   fin

3. écrire des procédures :

   procédure echanger **(E/S** val1,val2 : **Entier)**
   Déclaration temp : **Entier**
   début
   temp ← val1

val1 ← val2
val2 ← temp
fin

4. écrire des programmes :

Programme *exemple*
Déclaration entier1,entier2,entier3,min,max : **Entier**
fonction minimum2 **(**a,b : **Entier) : Entier**
...
fonction minimum3 **(**a,b,c : **Entier) : Entier**
...
procédure calculerMinMax3 **(E** a,b,c : **Entier** ; **S** min3,max3 : **Entier)**
début
min3 ← minimum3(a,b,c)
max3 ← maximum3(a,b,c)
fin
début
écrire(”Entrez trois entiers :”)
lire(entier1) ;
lire(entier2) ;
lire(entier3)
calculerMinMax3(entier1,entier2,entier3,min,max)
écrire(”la valeur la plus petite est ”,min,” et la plus grande est ”,max)
fin

5. décrire des types abstraits de donnée :

Nom: Complexe
Paramètre:
Utilise: **Réel**
Opérations:

Complexe:	**Réel** × **Réel** → Complexe
partieRéelle:	Complexe → **Réel**
partieImaginaire:	Complexe → **Réel**

Sémantiques:

Complexe:	l’opération qui permet de construire un complexe à partir de sa partie réelle et imaginaire
partieRéelle:	partieRéelle(a+bi)=a
partieImaginaire:	partieImaginaire(a+bi)=b

2 Utilisation

2.1 Inclusion

Pour pouvoir utiliser la package algorithme il faut l’inclure à son document LaTeX à l’aide de la commande `\usepackage` :

Listing 1 – Utilisation de usepackage

```
\usepackage{algorithme}
```

Attention : La package `algorithme.sty` requière la package `calc.sty`. Ce dernier doit donc être présent et référencé par votre système.

2.2 Environnement algorithme et tad

Lorsque l'on veut écrire un algorithme, une procédure, une fonction ou un programme, il faut utiliser l'environnement `algorithme`.

Listing 2 – Exemple d'utilisation de l'environnement algorithme

```
\begin{algorithme}
  \algo
  {\small}
  {somme}
  {Calculer la somme des n premiers entiers positifs , s=0+1+2+\ldots+n}
  {n : \naturel}
  {s : \naturel}
  {i : \naturel}
  {\affecter{s}{0}
    \pour {i} {0} {n} {} {\affecter{s}{s+i}}
  }
\end{algorithme}
```

Pour décrire un tad (voir page 11), il faut utiliser l'environnement `tad`.

Listing 3 – Exemple d'utilisation de l'environnement tad

```
\begin{tad}
  \tadNom{Complexe}
  \tadParametres{}
  \tadDependances{\reel}
  \begin{tadOperations}{partieImaginaire}
    \tadOperation{Complexe}{\tadDeuxParams{\reel}{\reel}}{\tadUnParam{Complexe}}
    \tadOperation{partieRéelle}{\tadUnParam{Complexe}}{\tadUnParam{\reel}}
    \tadOperation{partieImaginaire}{\tadUnParam{Complexe}}{\tadUnParam{\reel}}
  \end{tadOperations}
  \begin{tadSemantiques}{partieImaginaire}
    \tadSemantique{Complexe}{l'opération qui permet de construire un complexe
      à partir de sa partie réelle et imaginaire}
    \tadSemantique{partieRéelle}{partieRéelle(a+bi)=a}
    \tadSemantique{partieImaginaire}{partieImaginaire(a+bi)=b}
  \end{tadSemantiques}
\end{tad}
```

2.3 Algorithme, programme, procédure, fonction, etc.

2.3.1 Algorithme

Pour insérer un algorithme, il faut utiliser la commande `\algo`, qui prend en paramètre sept arguments :

1. La taille de la police de caractères (utilisation des commandes `\normalsize`, `\small`, etc.).
2. Le nom de l'algorithme
3. Son rôle
4. Les paramètres en entrée
5. Les paramètres en sortie
6. Les variables locales
7. Les instructions

Le listing 4 propose un exemple d'utilisation de la commande `\algo`.

Listing 4 – Exemple d'utilisation de la commande `\algo`

```
\begin{algorithme}
  \algo
  {\small}
  {somme}
  {Calculer la somme des n premiers entiers positifs, s=0+1+2+\ldots+n}
  {n : \naturel}
  {s : \naturel}
  {i : \naturel}
  {\affecter{s}{0}
    \pour {i} {0} {n} {} {\affecter{s}{s+i}}
  }
\end{algorithme}
```

2.3.2 Fonction

Pour insérer une fonction, il faut utiliser la commande `\fonction`, qui prend en paramètre cinq arguments :

1. Le nom de la fonction
2. Les paramètres
3. Le type de retour
4. Les variables locales
5. Les instructions

Dans la liste des instructions, il doit y avoir au moins une fois l'appel à la commande `\retourner` qui admet un argument en paramètre.

Le listing 5 propose un exemple d'utilisation de la commande `\fonction`.

Listing 5 – Exemple d'utilisation de la commande \fonction

```
\begin{algorithme}
  \small
  \fonction
  {abs}
  {unEntier : \entier}
  {\entier}
  {}
  {\sialorssinon{unEntier $\geq$ 0}
    {\retourner{unEntier}}
    {\retourner{-unEntier}}
  }
\end{algorithme}
```

2.3.3 Signature de fonction

Pour insérer une signature de fonction, il faut utiliser la commande \signaturefonction qui prend en paramètre trois arguments identiques aux trois premiers paramètres de la commande \fonction

2.3.4 Procédure

Pour insérer une procédure, il faut utiliser la commande \procedure, qui prend en paramètre quatre arguments :

1. Le nom de la procédure
2. Les paramètres
3. Les variables locales
4. Les instructions

Pour indiquer le passage de paramètre, on peut utiliser les commandes \paramEntree, \paramSortie et \paramEntreeSortie. Ces trois commandes prennent un seul argument.

Le listing 6 propose un exemple d'utilisation de la commande \procedure.

Listing 6 – Exemple d'utilisation de la commande \procedure

```
\begin{algorithme}
  \small
  \procedure
  {echanger}
  {\paramEntreeSortie{val1,val2 : \entier}}
  {temp : \entier}
  {\affecter{temp}{val1}
    \affecter {val1}{val2}
    \affecter {val2}{temp}
  }
\end{algorithme}
```

2.3.5 Signature de procédure

Tout comme pour les fonctions, on a la possibilité d'écrire une signature de procédure à l'aide de la commande `\signatureprocedure` qui prend en paramètre deux arguments identiques aux deux premiers arguments de la commande `\procedure`

2.3.6 Programme

Lorsque l'on veut présenter un programme dans sa globalité, on utilise la commande `\programme`. Cette commande admet quatre arguments en paramètre :

1. La taille des caractères (`\normalsize`, `\small`, etc.)
2. Le nom du programme
3. Les définitions locales aux programmes :
   - les constantes : utilisation de la commande `\constante` (voir page 9)
   - les types : utilisation de la commande `\type` (voir page 10)
   - les variables globales : utilisation de la commande `\variable`
4. Les instructions du programme

Le listing 7 propose un exemple d'utilisation de la commande `\programme`.

Listing 7 – Exemple d'utilisation de la commande `\programme`

```
\begin{algorithme}
  \programme
  {\small}
  {\textit{exemple}}
  {\variables{entier1,entier2,entier3,min,max : \entier}
    \signaturefonction{minimum2}{a,b : \entier}{\entier}\\
    \ldots
    \signaturefonction{minimum3}{a,b,c : \entier}{\entier}\\
    \ldots
    \procedure{calculerMinMax3}{\paramEntree{a,b,c : \entier};
      \paramSortie{min3,max3 : \entier}}
    {}
    {\affecter{min3}{minimum3(a,b,c)}
      \affecter{max3}{maximum3(a,b,c)}
    }
  }
  {\ecrire{"Entrez trois entiers :"}
    \lire{entier1};\lire{entier2};\lire{entier3}
    \instruction{calculerMinMax3(entier1,entier2,entier3,min,max)}
    \ecrire{"la valeur la plus petite est ",min," et la plus grande est ",max}
  }
\end{algorithme}
```

2.4 Instructions

Il existe 4 types d'instruction dans un algorithme :

1. les affectations
2. les tests
3. les boucles
4. les appels à des procédures

2.4.1 Affectation

Pour insérer une affectation on utilise la commande `\affecter` qui prend en paramètre deux arguments : ce qu'il y a à gauche du symbole d'affectation et ce qu'il y a droite.

Le listing 4 propose un exemple d'utilisation de la commande `\affecter`

2.4.2 Tests

Trois commandes sont proposées pour les tests : `\sialors`, `\sialorssinon` et `\cas`

La commande `\sialors` prend en paramètre deux arguments : la condition et les instructions. La commande `\sialorssinon` en ajoute un troisième pour les instructions de la partie sinon. Enfin, la commande `\cas` en prend deux, une pour le test et une pour les différents cas. Chaque cas est alors introduit par la commande `\casclausegenerale` qui prend en paramètre deux arguments : la ou les valeurs de la clause, et les instructions correspondantes. On a aussi la possibilité d'utiliser la clause *autre* via la commande `\casclauseautre` qui admet un seul paramètre : les instructions.

Le listing 5 propose un exmple d'utilisation de la commande `\sialorssinon`.

2.4.3 Boucles

Trois commandes sont proposées pour les tests :`\pour`, `\tantque` et `\repeter`.

La commande `\pour` prend cinq arguments en paramètre :

1. L'identifiant de boucle
2. La borne inférieure
3. La borne supérieure
4. Le pas (si ce paramètre est vide, le pas n'est pas affiché)
5. Les instructions

Le listing 4 propose un exemple d'utilisation de cette commande.

La commande `\tantque` prend deux arguments en paramètre :

1. La condition d'arrêt
2. Les instructions

Le listing 8 propose un exemple d'utilisation de cette commande.

Listing 8 – Exemple d'utilisation de la commande `\tantque`

```
\begin{algorithme}
  \algo
  {\small}
  {invFact}
  {Détermine le plus grand entier e telque e! $\leq$ n}
```

```
  {n : \naturel $\geq$ 1}
  {e : \naturel}
  {fact : \naturel}
  {\affecter{fact}{1}
    \affecter{e}{1}
    \tantque {fact $\leq$ n}
    {\affecter{e}{e+1}
      \affecter{fact}{fact*e}
    }
    \affecter{e}{e-1}
  }
\end{algorithme}
```

La commande \repeter prend deux arguments en paramètre :

1. Les instructions
2. La condition d'arrêt

Le listing 9 propose un exemple d'utilisation de cette commande.

Listing 9 – Exemple d'utilisation de la commande \repeter

```
\begin{algorithme}
  \algo
  {\small}
  {calculerPGCD}
  {Calculer le pgcd(a,b) à l'aide de l'algorithme d'euclide}
  {a,b : \naturel \non nul}
  {pgcd : \naturel}
  {reste : \naturel}
  {\repeter
    {\affecter{reste}{a mod b}
      \affecter{a}{b}
      \affecter{b}{reste}
    }{reste=0}
    \affecter{pgcd}{a}
  }
\end{algorithme}
```

2.4.4 Instructions

Lorsqu'une instruction est ni une affectation, ni un test, ni une boucle, il faut utiliser la commande \instruction. Le listing 7 présente un exemple d'utilisation de cette commande.

2.5 Commandes diverses

2.5.1 Les types de base

On peut utiliser les types de base à l'aide des commandes suivantes :
– \reel

- \entier
- \naturel
- \booleen
- \caractere
- \chaine

2.5.2 \ecrire

Cette commande permet d'utiliser l'instruction écrire. Elle admet un argument en paramètre.

2.5.3 \lire

Cette commande permet d'utiliser l'instruction lire. Elle admet un argument en paramètre.

2.5.4 \constante

Cette commande permet de créer une constante. Elle admet deux arguments en paramètre :

1. Le nom de la constante
2. La valeur de la constante

2.5.5 Les tableaux

Deux commandes sont disponibles pour créer des tableaux :

1. \tableauUneDimension qui permet de déclarer un tableau à une dimension. Cette commande admet trois arguments en paramètre :
   (a) L'intervalle (par exemple 1..MAX)
   (b) L'article possessif (de ou d')
   (c) Le type des éléments du tableau
2. \tableauDeuxDimensions qui permet de déclarer un tableau à deux dimensions. Cette commande admet quatre arguments en paramètre :
   (a) L'intervalle de la première dimension
   (b) L'intervalle de la deuxième dimension
   (c) L'article possessif
   (d) Le type des éléments du tableau

Le listing 10 propose un exmple d'utilisation de la commande \tableauUneDimension.

Listing 10 – Exemple d'utilisation de la commande \tableauUneDimension

```
\begin{algorithme}
  \algo
  {\tiny}
  {moyenne}
  {Affichage de la moyenne des notes
    d'une promo saisies par le prof}
  {}
```

```
  {}
  {somme,nbEleves,i : \naturel,
    lesNotes : \tableauUneDimension{1..100}{de }{\naturel}}
  {
    \affecter{somme}{0}
    \repeter
    {
      \ecrire{"Nombre d'eleves (maximum 100) :"}
      \lire{nbEleves}
    }
    {nbEleves>0 et nbEleves $\leq$ 100}
    \pour {i}{1}{nbEleves}{}
    {
      \ecrire{"Note de l'eleve numero ",i," :"}\\
      \lire{lesNotes[i]}
    }
    \pour {i}{1}{nbEleves}{}
    {
      \affecter{somme}{somme + lesNotes[i]}
    }
    \ecrire{"La moyenne est de :",somme/nbEleves}
  }
\end{algorithme}
```

2.5.6 \structure et \champstructure

Cette commande permet de créer des structures. Cette commande admet un argument en paramètre qui représente les attributs de cette structure à l'aide de la commande \champstructure.

Le listing 11 propose un exemple d'utilisation de ces deux commandes.

Listing 11 – Exemple d'utilisation de la commande \structure

```
\begin{algorithme}
    \type {Date} {\structure{
        \champstructure{leJour}{\entier}
        \champstructure{leMois}{\entier}
       \champstructure{lAnnee}{\entier}
      }
    }
\end{algorithme}
```

2.5.7 \type

Cette commande permet de créer des types. Elle admet deux arguments en paramètre :

1. Le nom du type a créer
2. Sa signification

Le listing 11 propose un exemple d'utilisation de cette commande.

2.6 Types abstraits de donnée (TAD)

On a aussi la possibilité de définir des types abstraits de données (que l'on retrouve dans la littérature sous les termes "types de données abstraits" ou "abstract datatype").

Comme le montre le listing 3, on encadre la définition d'un type abstrait de donnée par l'environnement `tad`. On utilise en suite les commandes et environnements suivants :

- la commande `\tadNom` qui permet de fixer le nom du TAD. Cette commande admet en paramètre un argument.
- la commande `\tadParametres` qui permet de fixer le paramètre de ce TAD, s'il existe. Cette commande admet en paramètre un argument.
- la commande `\tadDependances` qui permet d'identifier les types utilisés par le TAD. Cette commande admet en paramètre un argument.
- l'environnement `tadOperations` qui permet de lister, à l'aide de la commande `\tadOperation`, les opérations du TAD. Cet environnement admet un argument en paramètre optionnel : le nom de l'opération qui le a plus de caractères (permmetant d'aligner toutes les autres opérations sur cette dernière). La commande `\tadOperation` admet trois arguments en paramètre :
  1. le nom de l'opération
  2. le produit cartésien des types en entrée : il faut alors utiliser une des commandes suivantes :
     - `\tadUnParam` qui admet un argument en paramètre
     - `\tadDeuxParams` qui admet deux arguments en paramètre
     - `\tadTroisParams` qui admet trois arguments en paramètre
     - etc.
  3. le produit cartésien des types en sortie
- l'environnement `tadSemantiques` qui permet de lister, à l'aide de la commande `\tadSemantique`, la signification des opérations de ce TAD. Tout comme l'environnement `tadOperations`, cet environnement admet un argument en paramètre optionnel qui est aussi le nom de l'opération qui a le plus de caractères. La commande `\tadSemantique` admet deux arguments en paramètres :
  1. le nom de l'opération
  2. sa signification

3 Paramétrage

Tous les mots clefs utilisés par ce package sont modifiables à l'aide de la commande `\algoset`. Cette commande admet deux arguments en paramètre :

1. le nom de la commande correspondant au mot clef
2. le nouvelle valeur

Cette commande est utilisée avant le commencement du document (avant le `\begin{document}`).

Le listing 12 propose un exemple d'utilisation de cette commande qui permet de mettre les mot clefs "Entrée", "Sortie", "Entrée/Sortie" en lieu et place de "E","S", "E/S" lors des passage de paramètre en entrée, sortie et entrée/sortie dans les procédures.

Listing 12 – Exemple d'utilisation de la commande \algoset

```
\algoset{\motclefParamEntree}{Entrée}
\algoset{\motclefParamSortie}{Sortie}
\algoset{\motclefParamEntreeSortie}{Entrée/Sortie}
```

Les commandes correspondant aux mots clefs sont proposées par le tableau 1.

TAB. 1 – Commande et valeur pour les mots clefs

Commandes	Valeur par défaut
\motclefAlgoNom	Nom
\motclefAlgoRole	Role
\motclefAlgoEntree	Entrée
\motclefAlgoSortie	Sortie
\motclefAlgoDeclaration	Déclaration
\motclefAlgoDebut	début
\motclefAlgoFin	fin
\motclefEntier	Entier
\motclefNaturel	Naturel
\motclefReel	Réel
\motclefBooleen	Booléen
\motclefCaractere	Caratère
\motclefChaine	Chaîne de caractères
\Programme	Programme
\motclefRemarque	Remarque
\Type	Type
\motclefDeclaration	Déclaration
\motclefConstante	Constante
\motclefTableau	Tableau
\motclefStructure	Structure
\motclefFinstructure	fin structure
\motclefSi	si
\motclefAlors	alors
\motclefSinon	sinon
\motclefFinsi	finsi
\motclefCasou	cas où
\motclefCasouVaut	vaut
\motclefFincas	fincas
\motclefTantque	tant que
\motclefTantqueFaire	faire
\motclefFintantque	fintantque
\motclefRepeter	répéter
\motclefJusquaceque	jusqu'à ce que
\motclefPour	pour

Commandes	Valeur par défaut
\motclefPourA	à
\motclefPourPas	pas de
\motclefPourFaire	faire
\motclefFinPour	finpour
\motclefEcrire	écrire
\motclefLire	lire
\motclefFonction	fonction
\motclefRetourner	retourner
\motclefProcedure	procédure
\motclefParamEntree	E
\motclefParamSortie	S
\motclefParamEntreeSortie	E/S
\motclefTADNom	Nom
\motclefTADParametre	Paramètre
\motclefTADUtilise	Utilise
\motclefTADOperations	Opérations
\motclefTADSemantiques	Sémantiques