Package `algo`

Vincent Lozano

Version 1.4 — Février 2008

http://lozzone.free.fr

Résumé

Ce document décrit l'utilisation et la configuration du merveilleux package `algo` permettant d'écrire des algorithmes dans un document LaTeX. Le style est celui d'un langage typé se rapprochant du Pascal ou du C. On peut d'ailleurs utiliser l'une ou l'autre des syntaxes pour les déclarations de variables et de types. Ce package est entièrement paramétrable et devrait s'adapter à la plupart des besoins pour ce qui est de l'écriture des algorithmes. Il a été créé pour produire des supports de cours d'informatique dispensé aux élèves de l'École nationale d'ingénieurs de Saint-Étienne. Cette version, programmée avec les pieds, est la première diffusion, seuls cinq utilisateurs ont vaillement passé les tests...

```
Si vous voyez ce que je veux dire Alors
| Tournez la page
Sinon
| Répéter
| | Relisez le résumé
| Jusque ce que vous ayez compris
```

Sommaire

1 Introduction **1**

2 Utilisation **2**

3 Personnalisations **12**

4 Algorithmes flottants **19**

5 Retour **20**

1 Introduction

Ce package permet d'écrire des programmes en langage algorithmique. Sa syntaxe est vaguement inspirée du package `algorithmic` de Peter `Williams`. Pour l'utiliser, il suffit d'insérer la commande :

```
\usepackage{algo}
```

dans le préambule du document. Les options qu'il est possible de passer au package sont :

- `noendkeyword` qui supprime les mots de fin de clause (cf. 3.2) ;
- `numbered` pour numéroter les algorithmes (cf. 3.1) ;
- `english` pour écrire les mots clé en anglais ;
- `cstyledecls` pour utiliser le style du langage C pour les déclarations et les arguments de fonction (cf. § 3.10 page 18).

Après chargement du package, on dispose d'un environnement `algo` dans lequel des commandes spécifiques les programmes en langage algorithmique :

```
\begin{algo}
  ...
\end{algo}
```

2 Utilisation

Tout d'abord, le fameux algorithme « hello world » :

```
\begin{algo}
  \ALGO{bonjour monde}
  \VAR
  \DECLVAR{c}{chaine}
  \DECLVAR[compteur]{i}{entier}
  \ENDVAR
  \BEGIN
  \STATE{c \recoit{} "bonjour monde"}
  \COMMENT{on affiche 3 fois le message}
  \FOR{i}{1}{3}
  \STATE{Afficher(c)}
  \ENDFOR
  \END
\end{algo}
```

Algorithme bonjour monde
Variable
  c : *chaine*
  i : *entier* { *compteur* }
Début
  c ← "bonjour monde"
  { *on affiche 3 fois le message* }
  Pour i **variantDe** 1 **à** 3 **Faire**
    Afficher(c)
  FinPour
Fin

On constatera que tout est produit dans la fonte « machine à écrire » sauf les chaînes de caractères (délimitée par le caractère ") en roman, les commentaires en roman penché et les types en italique (cf. § 3.8 page 15 pour voir comment changer cela). On pourra dores et déjà noter que l'environnement `algo` ne tient pas compte des espaces et sauts de ligne, par conséquent le même algorithme aurait été produit par le code :

```
  \begin{algo}
  \ALGO{bonjour monde}
  \VAR \DECLVAR{c}{chaine} \DECLVAR{i}{entier}\ENDVAR
  \BEGIN
  \STATE{c \recoit{} "bonjour monde"}
  \FOR{i}{1}{3}\STATE{Afficher(c)} \ENDFOR
  \END
  \end{algo}
```

2.1 Constantes, types et Variables

On pourra déclarer les constantes en utilisant des blocs :

```
\CONST  \ENDCONST

\TYPE  \ENDTYPE

\VAR  \ENDVAR
```

Constante
Type
Variable

2.1.1 Constantes

À l'intérieur de chacun de ces blocs, on pourra déclarer des constantes en utilisant trois syntaxes :
– `\DECLCONST{`⟨*nom*⟩`}{`⟨*val*⟩`}` :

Constante
  ⟨*nom*⟩ = ⟨*val*⟩

– `\DECLCONST[`⟨*description*⟩`]{`⟨*nom*⟩`}{`⟨*val*⟩`}` :

Constante
  ⟨*nom*⟩ = ⟨*val*⟩ { ⟨*description*⟩ }

– `\DECLCONST[`⟨*description*⟩`][`⟨*type*⟩`]{`⟨*nom*⟩`}{`⟨*val*⟩`}` :

Constante
| ⟨*nom*⟩ : ⟨*type*⟩=⟨*val*⟩ { ⟨*description*⟩ }

L'argument optionnel ⟨*description*⟩ permet d'ajouter une description sous la forme d'un commentaire à côté de la déclaration, et le deuxième argument optionnel permet de préciser le type de la constante si nécessaire :

```
\CONST
\DECLCONST[][booléen]{OUVERT}{Vrai}
\DECLCONST[$\pi$][flottant]{PI}{3.14}
\DECLCONST[40 trucs au +]{MAX\_BIDULE}{40}
\ENDCONST
```

Constante
| OUVERT : *booléen*=Vrai
| PI : *flottant*=3.14 { π }
| MAX_BIDULE = 40 { *40 trucs au* + }

2.1.2 Variables

Les variables peuvent être définies avec la syntaxe :

`\DECLVAR{`⟨*nom*⟩`}{`⟨*type*⟩`}` qui donnera :

Variable
| ⟨*nom*⟩ : ⟨*type*⟩

ou, si l'on veut ajouter un commentaire à la déclaration de la variable :

`\DECLVAR[`⟨*desc*⟩`]{`⟨*nom*⟩`}{`⟨*type*⟩`}` qui donnera :

Variable
| ⟨*nom*⟩ : ⟨*type*⟩ { ⟨*desc*⟩ }

Quelques exemples :

```
\VAR
\DECLVAR[compteurs]{i,j}{entier}
\DECLVAR[le machin]{res}{flottant}
\ENDVAR
```

Variable
| i,j : *entier* { *compteurs* }
| res : *flottant* { *le machin* }

2.1.3 Types

On peut définir un nouveau type en écrivant :

```
\TYPE
\DECLTYPE{Tbidule}{entier}
\ENDTYPE
```

Type
| *Tbidule* = *entier*

L'environnement `algo` permet également d'écrire la définition des types structurés tableau et enregistrement. La définition de type tableau peut être produite de la manière suivante :

```
\TYPE
\DECLTYPE{Ttab}{\ARRAY{entier}}
\ENDTYPE
```

Type
| *Ttab* = **Tableau** *entier* []

On pourra utiliser des tableaux à deux dimensions comme suit :

```
\TYPE
\DECLTYPE{Tmatrice}{%
   \ARRAY[2]{flottant}}
\ENDTYPE
```

Type
| *Tmatrice* = **Tableau** *flottant* [] []

Pour utiliser des tableaux dont on fixe la taille, on pourra avoir recours à la fonction `\DIMARRAY` prenant jusqu'à trois arguments optionnels :

```
\TYPE
\DECLTYPE{Tmatrice}{%
   \DIMARRAY[5][7]{flottant}}
\ENDTYPE
\VAR
\DECLVAR{montab}{\DIMARRAY[4]{entier}}
\ENDVAR
```

Type
| *Tmatrice* = **Tableau** *flottant* [5][7]
Variable
| montab : **Tableau** *entier* [4]

Pour ce qui concerne les enregistrements, on utilisera la syntaxe suivante :

```
\TYPE
\RECORD{Tsequence}
\DECLFIELD{nbre}{entier}
\DECLFIELD{donnees}{%
  \DIMARRAY[20]{flottant}}
\ENDRECORD
\ENDTYPE
```

Type
| *Tsequence* = **Enregistrement**
| | nbre : *entier*
| | donnees : **Tableau** *flottant* [20]
| **FinEnregistrement**

2.2 Algorithmes et instructions

On écrit un algorithme dans un bloc :

```
\BEGIN

\END
```

Début
Fin

et les instructions à l'aide de la clause :

`\STATE{...}`

Par exemple :

```
\VAR
\DECLVAR{i,j}{entier}
\DECLVAR{bidule}{flottant}
\ENDVAR
\BEGIN
\STATE{j \recoit{} 20}
\STATE{i \recoit{} 2*j}
\END
```

Variable
| i,j : *entier*
| bidule : *flottant*
Début
| j ← 20
| i ← 2*j
Fin

2.3 Structures de contrôle

Le package propose plusieurs structures de contrôle dont voici quelques exemples pour en expliciter les constructions.

2.3.1 Si .. Alors ... Sinon

La fameuse structure de contrôle sans la clause sinon :

```
\VAR
\DECLVAR{i}{entier}
\ENDVAR
\BEGIN
\IF{i=0}
\STATE{Afficher("i est nul")}
\ENDIF
\END
```

Variable
  i : *entier*
Début
  Si i=0 **Alors**
    Afficher("i est nul")
  FinSi
Fin

La même avec la clause sinon :

```
\VAR
\DECLVAR{b}{booléen}
\ENDVAR
\BEGIN
\IF{b=VRAI}
\STATE{Afficher("Oui"}
\ELSE
\STATE{Afficher("Non")}
\ENDIF
\END
```

Variable
  b : *booléen*
Début
  Si b=VRAI **Alors**
    Afficher("Oui")
  Sinon
    Afficher("Non")
  FinSi
Fin

On peut également construire une clause du type alternative multiple de la manière suivante :

```
\VAR
\DECLVAR{i}{entier}
\ENDVAR
\BEGIN
\IF{i=0}
\STATE{afficher("Nul")}
\ELSEIF{i=1}
\STATE{afficher("Un")}
\ELSEIF{i=2}
\STATE{afficher("Deux")}
\ELSE
\STATE{afficher("Ni 0, ni 1, ni 2")}
\ENDIF
\END
```

Variable
  i : *entier*
Début
  Si i=0 **Alors**
    afficher("Nul")
  SinonSi i=1 **Alors**
    afficher("Un")
  SinonSi i=2 **Alors**
    afficher("Deux")
  Sinon
    afficher("Ni 0, ni 1, ni 2")
  FinSi
Fin

2.3.2 SelonQue ...

```
\VAR
\DECLVAR{rep}{entier}
\ENDVAR
\BEGIN
\STATE{Afficher("Votre choix ?")}
\STATE{Lire(rep)}
\SWITCH{rep}
\CASE{1}
\STATE{bidule(rep)}
\ENDCASE
\CASE{2}
\STATE{truc(rep-1)}
\ENDCASE
\CASE{3}
\STATE{chouette(2*rep)}
\ENDCASE
\ENDSWITCH
\END
```

```
Variable
    rep : entier
Début
    Afficher("Votre choix ?")
    Lire(rep)
    SelonQue rep vaut
        Cas 1 :
            bidule(rep)
        FinCas
        Cas 2 :
            truc(rep-1)
        FinCas
        Cas 3 :
            chouette(2*rep)
        FinCas
    FinSelonQue
Fin
```

On peut également écrire un Selon Que avec un cas par défaut :

```
\SWITCH{rep}
\CASE{1}
\STATE{bidule(rep)}
\ENDCASE
\CASE{2}
\STATE{truc(rep-1)}
\ENDCASE
\DEFAULT
\STATE{par\_defaut(2*rep)}
\ENDCASE
\ENDSWITCH
```

```
SelonQue rep vaut
    Cas 1 :
        bidule(rep)
    FinCas
    Cas 2 :
        truc(rep-1)
    FinCas
    Sinon
        par_defaut(2*rep)
    FinCas
FinSelonQue
```

Enfin on peut utliser une version condensée de la clause **Cas** :

```
\SWITCH{rep}
\SHORTCASE{1}{bidule(rep)}
\SHORTCASE{2}{truc(rep-1)}
\DEFAULT
\STATE{par\_defaut(2*rep)}
\ENDCASE
\ENDSWITCH
```

```
SelonQue rep vaut
    Cas 1 : bidule(rep)
    Cas 2 : truc(rep-1)
    Sinon
        par_defaut(2*rep)
    FinCas
FinSelonQue
```

2.3.3 Répéter ... Jusqu'à

```
\VAR
\DECLVAR{i}{entier}
\ENDVAR
\BEGIN
\STATE{i\recoit{}0}
\REPEAT
\STATE{Afficher("2 x ",i," = ",2*i)}
\STATE{i\recoit{}i+1}
\ENDREPEAT{i>10}
\END
```

Variable
  i : *entier*
Début
  i ← 0
  Répéter
    Afficher("2 x ",i," = ",2*i)
    i ← i+1
  Jusque i>10
Fin

2.3.4 Répéter ... TantQue

```
\VAR
\DECLVAR{rep}{caractère}
\ENDVAR
\BEGIN
\REPEAT
\STATE{Afficher("Alors ? (o/n)")}
\STATE{Lire(rep)}
\ENDREPEAT[while]{rep$\not =$'o'
                  ET rep$\not=$'n'}
\END
```

Variable
  rep : *caractère*
Début
  Répéter
    Afficher("Alors ? (o/n)")
    Lire(rep)
  TantQue rep≠'o' ET rep≠'n'
Fin

Notez l'utilisation de l'argument optionnel `while` à la commande `\ENDREPEAT`.

2.3.5 TantQue .. Faire

```
\VAR
\DECLVAR{rep}{caractère}
\ENDVAR
\BEGIN
\STATE{rep\recoit{}'?'}
\WHILE{rep$\not=$'o' ET
       rep$\not=$'n'}
\STATE{Afficher("Alors ? (o/n)")}
\STATE{Lire(rep)}
\ENDWHILE
\END
```

Variable
  rep : *caractère*
Début
  rep ← '?'
  TantQue rep≠'o' ET rep≠'n' **Faire**
    Afficher("Alors ? (o/n)")
    Lire(rep)
  FinTantQue
Fin

2.3.6 Boucle Pour ...

```
\VAR
\DECLVAR{c}{entier}
\ENDVAR
\BEGIN
\STATE{Afficher("La table de 3 : ")}
\FOR{c}{1}{10}
\STATE{Afficher("3 x ",c," = ",3*c)}
\ENDFOR
\END
```

Variable
  c : *entier*
Début
  Afficher("La table de 3 : ")
  Pour c **variantDe** 1 **à** 10 **Faire**
    Afficher("3 x ",c," = ",3*c)
  FinPour
Fin

On peut également utiliser la notion de pas d'incrémentation avec la commande \FORSTEP :

```
\VAR
\DECLVAR{c}{entier}
\ENDVAR
\BEGIN
\FORSTEP{c}{1}{10}{2}
\STATE{Afficher(c)}
\ENDFOR
\END
```

Variable
  c : *entier*
Début
  Pour c **variantDe** 1 10 **parPasDe** 2 **Faire**
    Afficher(c)
  FinPour
Fin

Pour avoir une version générale de la boucle **Pour**, on utilisera la commande \FORGEN

```
\BEGIN
\FORGEN{chaque expression $e$}
\STATE{décoder $e$}
\STATE{exécuter $e$}
\ENDFOR
\END
```

Début
  Pour chaque expression e **Faire**
    décoder e
    exécuter e
  FinPour
Fin

Enfin il existe une version boucle **PourChaque** :

```
\BEGIN
\FOREACH[clef]{val}{montab}
\STATE{Faire\_truc\_avec(clef,val)}
\ENDFOREACH
\END
```

Début
  PourChaque clef ▷ val **De** montab **Faire**
    Faire_truc_avec(clef,val)
  FinPourChaque
Fin

ou plus simplement :

```
\BEGIN
\FOREACH{val}{montab}
\STATE{Faire\_truc\_avec(val)}
\ENDFOREACH
\END
```

Début
  PourChaque val **De** montab **Faire**
    Faire_truc_avec(val)
  FinPourChaque
Fin

2.3.7 Contrôle de l'exécution

On pourra indiquer qu'on veut explicitement arrêter le déroulement de l'algorithme avec la commande \EXIT produisant par défaut le mot clé **Stop** :

```
\BEGIN
\IF{Plus de mémoire}
\STATE{Afficher("Bon bé...")}
\EXIT[Arrêt du programme]
\ENDIF
\END
```

Début
  Si Plus de mémoire **Alors**
    Afficher("Bon bé...")
    Stop { *Arrêt du programme* }
  FinSi
Fin

Enfin, on pourra indiquer qu'on veut explicitement quitter le déroulement de la structure de contrôle itérative courante avec la commande \BREAK produisant par défaut le mot clé **Quitter** :

```
\BEGIN
\FORGEN{chaque bouteille B}
\STATE{Déboucher(B)}
\IF{Si B est bouchonné}
\STATE{Contacter(révendeur)}
\BREAK[sortie de la boucle Pour]
\ENDIF
\ENDFOR
\END
```

Début
  Pour chaque bouteille B **Faire**
    Déboucher(B)
    Si Si B est bouchonné **Alors**
      Contacter(revendeur)
      Quitter { *sortie de la boucle Pour* }
    FinSi
  FinPour
Fin

2.4 Routines, procédures et autre fonctions

Le package `algo` permet de construire une routine de la manière suivante :

\FUNCTION{⟨*identificateur*⟩}{⟨*arguments*⟩}{ ⟨*type renvoyé*⟩}

S'il est nécessaire de faire explicitement la distinction entre procédure et fonction comme dans le langage Pascal, on pourra utiliser la syntaxe suivante :

\PROC{⟨*identificateur*⟩}{⟨*arguments*⟩}

pour une procédure, et pour une fonction :

\FUNC{⟨*identificateur*⟩}{⟨*arguments*⟩}{⟨*type renvoyé*⟩}

Où ⟨*arguments*⟩ pourra être un ou plusieurs appels aux commandes suivante :
- \pfarg{⟨*id*⟩}{⟨*identificateur*⟩} pour un argument ;
- \pfargin{⟨*id*⟩}{⟨*identificateur*⟩} pour un argument de nature « donnée » ;
- \pfargout{⟨*id*⟩}{⟨*identificateur*⟩} pour un argument de nature « résultat » ;
- \pfarginout{⟨*id*⟩}{⟨*identificateur*⟩} pour un argument de nature « transformée ».

ces commandes produiront le noms des arguments formels et leur type en respectant la cohérence de l'ensemble des algorithmes. Voici maintenant quelques exemples.

```
\FUNCTION{message}{\pfarg{c}{chaine}}{}
\VAR
\DECLVAR{i}{entier}
\ENDVAR
\BEGIN
\FOR{i}{1}{10}
\STATE{Afficher("Message : ",c)}
\ENDFOR
\END
```

Routine message(c : *chaine*)
Variable
  i : *entier*
Début
  Pour i **variantDe** 1 **à** 10 **Faire**
    Afficher("Message : ",c)
  FinPour
Fin

```
\FUNCTION{somme}{%
   \pfarg{x,y}{entier}}{entier}
\BEGIN
\RETURN{x+y}
\END
```

Routine somme(x,y : *entier*) : *entier*
Début
  Retourner x+y
Fin

```
\PROC{pond}{
   \pfargin{x,y}{entier} ;
   \pfargout{s}{flottant}}
\BEGIN
\STATE{s \recoit{} .3*x+.7*y}
\END
```

Procédure pond(Ⓓ x,y : *entier* ; Ⓡ s : *flottant*)
Début
  s ← .3*x+.7*y
Fin

```
\FUNC{somme}{%
   \pfarg{x,y}{entier}}{entier}
\BEGIN
\RETURN{x+y}
\END
```

Fonction somme(x,y : *entier*) : *entier*
Début
  Retourner x+y
Fin

2.5 Commentaires

On peut insérer un commentaire dans le code grâce à la commande \COMMENT :

```
\BEGIN
\COMMENT{Début de l'algo}
\END
```

Début
  { *Début de l'algo* }
Fin

Certaines structures de contrôle prennent en paramètre optionnel un commentaire qui sera inséré sur la même ligne. Ces structures sont :

```
\DECLVAR \DECLTYPE \DECLCONST \DECLFIELD \STATE \IF \ELSE \ELSEIF
\FOR \WHILE \REPEAT \SWITCH \CASE \DEFAULT \FUNCTION \FUNC \PROC
\RETURN \BREAK \EXIT
```

Voici un exemple :

```
\begin{algo}
  \VAR
  \DECLVAR[compteur]{i}{entier}
  \ENDVAR
  \BEGIN
  \COMMENT{une super boucle}
  \FOR[on compte jusqu'à 10]{i}{1}{10}
  \IF[test de parité]{i mod 2 = 0}
  \STATE[affichage à l'écran]{Afficher(i," pair")}
  \ELSE[là c'est impair]
  \STATE{Afficher(i," impair")}
  \ENDIF
  \ENDFOR
  \END
\end{algo}
```

qui donne :

```
Variable
  i : entier   { compteur }
Début
  { une super boucle }
  Pour i variantDe 1 à 10 Faire   { on compte jusqu'à 10 }
    Si i mod 2 = 0 Alors   { test de parité }
      Afficher(i," pair")   { affichage à l'écran }
    Sinon   { là c'est impair }
      Afficher(i," impair")
    FinSi
  FinPour
Fin
```

2.6 Algorithmes en plusieurs morceaux

De manière à pouvoir reprendre un algorithme interrompu, on peut utiliser l'option `continue` :

```
bon bé ouala :
\begin{algo}
  \BEGIN
  \FOR{i}{0}{9}
  \IF{i mod 2 = 0}
  \STATE{Afficher(i," pair")}
\end{algo}
alors oui au fait :
\begin{algo}[continue]
  \ELSE
  \STATE{Afficher(i," impair")}
  \ENDIF
  \ENDFOR
  \END
\end{algo}
```

bon bé ouala :

```
Début
  Pour i variantDe 0 à 9 Faire
    Si i mod 2 = 0 Alors
      Afficher(i," pair")
```

alors oui au fait :

```
    Sinon
      Afficher(i," impair")
    FinSi
  FinPour
Fin
```

2.7 Écrire en langage alorithmique dans le texte

Le package `algo` propose cinq commandes pour pouvoir mentionner des « bouts » d'algorithmes dans le texte. Ces cinq commandes sont :

- `\textalgotxt{`⟨*texte*⟩`}` pour produire du texte comme le corps d'un algorithme ;
- `\textalgostr{`⟨*texte*⟩`}` pour produire du texte comme une chaîne de caractères ;
- `\textalgocom{`⟨*texte*⟩`}` pour produire du texte comme un commentaire ;
- `\textalgotype{`⟨*texte*⟩`}` pour produire du texte comme un type de données ;
- `\textalgokw{`⟨*mot clef*⟩`}` pour écrire un mot clef.

Voici un exemple :

```
Dans la boucle \textalgokw{for},
on utilise la  variable~\textalgotxt{i}
de type \textalgotype{entier}.
```

Dans la boucle **Pour**, on utilise la variable `i` de type *entier*.

2.8 Algorithme informel

Il est possible grâce à l'option `informal` de l'environnement `algo` d'écrire un algorithme « informel ». Il s'agit juste d'une manipulation commode permettant de passer en police roman et d'enlever la bordure de la boîte :

```
\begin{algo}[informal]
  \BEGIN
  \STATE{Demander un truc à l'utilisateur}
  \STATE{Calculer les machins}
  \STATE{Afficher le bazar}
  \END
\end{algo}
```

Début
| Demander un truc à l'utilisateur
| Calculer les machins
| Afficher le bazar

Fin

3 Personnalisations

On peut modifier l'allure de l'environnement `algo` en agissant sur plusieurs catégories de paramètres. Il existe trois manières de modifier l'environnement :

1. en passant des options au package (cf. § 1) ;
2. pour *toutes* les occurrences de l'environnement, on redéfinira dans le préambule du document, des commandes et des longueurs ;
3. pour *une* occurrence particulière, on passera des arguments optionnels à l'environnement `algo`.

3.1 Numérotation et référence

On peut numéroter les lignes de l'algorithme en utilisant l'option `numbered` :

```
\begin{algo}[numbered]
  \BEGIN
  \FOR{i}{0}{9}
  \IF{i mod 2 = 0}
  \STATE{Afficher(i," pair")}
  \ELSE
  \STATE{Afficher(i," impair")}
  \ENDIF
  \ENDFOR
  \END
\end{algo}
```

```
1 Début
2   Pour i variantDe 0 à 9 Faire
3     Si i mod 2 = 0 Alors
4       Afficher(i," pair")
5     Sinon
6       Afficher(i," impair")
7     FinSi
8   FinPour
9 Fin
```

Par défaut, les algorithmes ne sont pas numérotés. On peut changer ce comportement en passant l'option `numbered` au package :

```
\usepackage[numbered]{algo}
```

On peut également utiliser n'importe où dans le document, les commandes :
- `\numberedalgo` qui enclenche la numérotation des algos ;
- `\unnumberedalgo` qui débraye cette numérotation.

Enfin, on notera qu'il est possible de faire référence aux lignes d'un algorithme comme dans l'exemple suivant :

```
À la ligne~\ref{truc} de l'algo [...]
L'instruction~\textalgokw{return}
de la ligne~\ref{leretour} permet de
renvoyer la valeur \textalgotxt{4.5}.
\begin{algo}[numbered]
  \FUNC{f}{\pfarg{y}{entier}}{flottant}
  \VAR
  \DECLVAR{i}{entier}  \label{truc}
  \ENDVAR
  \BEGIN
  \STATE{hop là}
  \RETURN{4.5}\label{leretour}
  \END
\end{algo}
```

À la ligne 3 de l'algo [...] L'instruction **Retourner** de la ligne 6 permet de renvoyer la valeur `4.5`.

```
1 Fonction f( y : entier ) : flottant
2 Variable
3   i : entier
4 Début
5   hop là
6   Retourner 4.5
7 Fin
```

3.2 Mot clef de fin de clause

Dans certaines circonstances, on peut vouloir se passer des mots clef de fin de clause, on écrira alors :

```
\begin{algo}[noendkeyword]
  \BEGIN
  \FOR{i}{0}{9}
  \IF{i mod 2 = 0}
  \STATE{Afficher(i," pair")}
  \ELSE
  \STATE{Afficher(i," impair")}
  \ENDIF
  \ENDFOR
  \END
\end{algo}
```

Début
  Pour i **variantDe** 0 **à** 9 **Faire**
    Si i mod 2 = 0 **Alors**
      Afficher(i," pair")
    Sinon
      Afficher(i," impair")
Fin

Les mots clef concernés sont :

```
\ENDIF \ENDWHILE \ENDFOR \ENDFORGEN \ENDCASE
\ENDSWITCH \ENDRECORD
```

Pour supprimer, ou activer les mots clé de fin de clause pour tous les environnements `algo` on utilisera respectivement les commandes :

- `\noalgoendkeyword`, et :
- `\algoendkeyword`.

Notez que les mots clé de fin de clause sont activés par défaut ; on pourra les désactiver par défaut en utilisant l'option de package :

```
\usepackage[noendkeyword]{algo}
```

dans ce cas, pour les réactiver momentanément on utilisera l'option `endkeyword` comme argument à l'environnement.

3.3 Boîte englobante

La boîte qui englobe l'environnement peut être modifiée en agissant sur l'épaisseur du trait et l'espace qui sépare l'algo lui-même du cadre de la boîte. On peut tout d'abord supprimer le cadre de la manière suivante :

```
\begin{algo}[noframe]
\BEGIN
\FORGEN{chaque expression $e$}
\STATE{décoder $e$}
\STATE{exécuter $e$}
\ENDFORGEN
\END
\end{algo}
```

Début
  Pour chaque expression e **Faire**
    décoder e
    exécuter e
  FinPour
Fin

Et changer le trait du cadre comme suit :

```
\begin{algo}[boxsep=2pt,
             boxrule=2pt]
\BEGIN
\FORGEN{chaque expression $e$}
\STATE{décoder $e$}
\STATE{exécuter $e$}
\ENDFORGEN
\END
\end{algo}
```

Début
  Pour chaque expression e **Faire**
    décoder e
    exécuter e
  FinPour
Fin

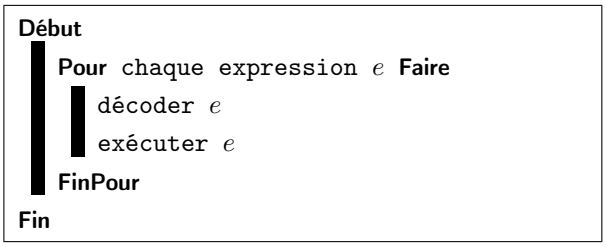
Pour modifier les boîtes englobantes de tous les environnements `algo`, on agira sur les longueurs `\algoboxrule` et `\algoboxsep`. Pour information, sont définies par défaut :

```
\setlength{\algoboxrule}{.4pt}
\setlength{\algoboxsep}{5pt}
```

3.4 Barre verticale

On peut imposer une largeur de barre :

```
\begin{algo}[rulewidth=5pt]
\BEGIN
\FORGEN{chaque expression $e$}
\STATE{décoder $e$}
\STATE{exécuter $e$}
\ENDFORGEN
\END
\end{algo}
```



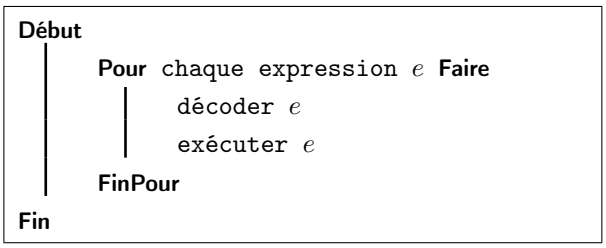
Pour modifier les barres verticales de tous les environnements `algo`, on agira sur la longueur `\algorulewidth`. Pour information, est définie par défaut :

```
\setlength{\algorulewidth}{1pt}
```

3.5 Indentation

On peut changer la largeur des indentations de chacun des blocs :

```
\begin{algo}[indentwidth=30pt]
\BEGIN
\FORGEN{chaque expression $e$}
\STATE{décoder $e$}
\STATE{exécuter $e$}
\ENDFORGEN
\END
\end{algo}
```



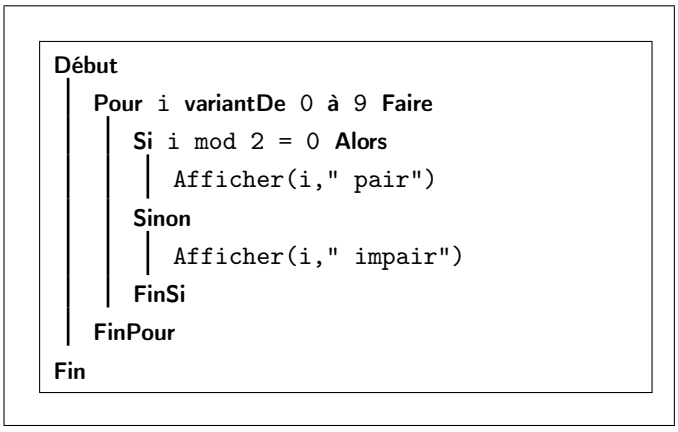
Pour modifier la largeur des blocs de tous les environnements `algo`, on agira sur la longueurs `\algoindentwidth`. Pour information, est définie par défaut :

```
\setlength{\algoindentwidth}{15pt}
```

3.6 Marges

On peut régler les marges gauche, droite, haute et basse de l'environnement `algo` :

```
\begin{algo}[rightmargin=5pt,
             leftmargin=10pt,
             topmargin=7pt,
             bottommargin=2pt]
  \BEGIN
  \FOR{i}{0}{9}
  \IF{i mod 2 = 0}
  \STATE{Afficher(i," pair")}
  \ELSE
  \STATE{Afficher(i," impair")}
  \ENDIF
  \ENDFOR
  \END
\end{algo}
```



Pour modifier les quatre marges de tous les environnements `algo`, on agira sur les quatre longueurs suivantes qui sont définies par défaut comme suit :

```
\setlength{\algoleftmargin}{15pt}
\setlength{\algorightmargin}{10pt}
\setlength{\algotopmargin}{6pt}
\setlength{\algobottommargin}{6pt}
```

TABLE 1 – mots clef

commande	mot clé	commande	mot clé
\algokeywordalgo	Algorithme	\algokeywordbegin	Début
\algokeywordend	Fin	\algokeywordif	Si
\algokeywordwhile	TantQue	\algokeywordendwhile	FinTantQue
\algokeywordrepeat	Répéter	\algokeyworduntil	Jusqu'à
\algokeywordendif	FinSi	\algokeywordelse	Sinon
\algokeywordelseif	SinonSi	\algokeyworddo	Faire
\algokeywordthen	Alors	\algokeywordfor	Pour
\algokeywordendfor	FinPour	\algokeywordfrom	variantDe
\algokeywordto	à	\algokeywordswitch	SelonQue
\algokeywordchoix	Choix	\algokeywordendchoix	FinChoix
\algokeywordendswitch	FinSelonQue	\algokeywordvaut	vaut
\algokeywordcase	Cas	\algokeywordendcase	FinCas
\algokeyworddefault	Sinon	\algokeywordvar	Variable
\algokeywordconst	Constante	\algokeywordtype	Type
\algokeywordproc	Procédure	\algokeywordfunc	Fonction
\algokeywordreturn	Renvoyer	\algokeywordarray	Tableau
\algokeywordexit	Stop	\algokeywordbreak	Quitter

3.7 Mots clé

Les mots clé des algorithmes sont produits par les commandes du tableau 1. Toutes celles-ci peuvent être redéfinies par l'intermédiaire de la commande \renewcommand. On pourra par exemple écrire :

```
\renewcommand{\algokeywordendif}{iS}
\renewcommand{\algokeywordendfor}{rouP}
\begin{algo}
\BEGIN
\FOR{i}{0}{10}
\IF{g(i)>4}
\STATE{afficher("hop là")}
\ENDIF
\ENDFOR
\END
\end{algo}
```

```
Début
    Pour i variantDe 0 à 10 Faire
        Si g(i)>4 Alors
            afficher("hop là")
        iS
    rouP
Fin
```

Par commodité, on peut passer l'option english au package algo :

```
\usepackage[english]{algo}
```

pour disposer des mots clé en anglais. Par ailleurs, les commandes \algofrenchkeywords et \algoenglishkeywords permettent de passer dans un même document de l'anglais au français. Voici un exemple idiot illustrant l'utilisation de l'une de ces commandes :

```
\begin{algo}
\BEGIN
\FORGEN{chaque expression $e$}
\STATE{décoder $e$}
\algoenglishkeywords
\STATE{exécuter $e$}
\ENDFORGEN
\END
\end{algo}
```

```
Début
    Pour chaque expression e Faire
        décoder e
        exécuter e
    EndFor
End
```

3.8 Fontes

Quatre commandes sont définies pour produire les différents caractères de l'environnement algo :
- \algotextfont pour le texte ;
- \algocommentfont pour les commentaires ;

– `\algostringfont` pour les chaînes ;
– `\algokeywordfont` pour les mots clefs.

Par défaut, ces commandes sont définies par :

```
\newcommand{\algotextfont}{\ttfamily\upshape}
\newcommand{\algostringfont}{\rmfamily\upshape}
\newcommand{\algocommentfont}{\rmfamily\slshape}
\newcommand{\algotypefont}{\rmfamily\itshape}
\newcommand{\algokeywordfont}[1]{{\small\sffamily\upshape\bfseries#1}}
```

Le texte de l'algo est donc en famille « machine à écrire », les chaînes de caractères en roman, les types en roman italique et les commentaires en roman penché. Les mots clé sont quant à eux produits dans la famille sans sérif, en gras et en petite taille. Attention, `\algokeywordfont` est une commande, alors que les quatre premières sont des déclarations. On peut donc changer l'allure d'un algorithme en définissant par exemple :

```
\renewcommand{\algotextfont}{\rmfamily}
\renewcommand{\algostringfont}{\ttfamily}
\renewcommand{\algocommentfont}{\rmfamily\itshape}
\renewcommand{\algokeywordfont}[1]{\uline{\sffamily\slshape\mdseries#1}\/}
```

un algorithme ressemblera à ce qui suit : [1]

```
Début
  { une super boucle }
  Pour i variantDe 0 à 9 Faire
    Si i mod 2 = 0 Alors
      Afficher(i," pair")
    Sinon   { là c'est impair }
      Afficher(i," impair")
    FinSi
  FinPour
Fin
```

Un autre exemple—immonde—avec :

```
\renewcommand{\algotextfont}{\slshape}
\renewcommand{\algostringfont}{\ttfamily\upshape}
\renewcommand{\algocommentfont}{\rmfamily\upshape\bfseries}
\renewcommand{\algokeywordfont}[1]{%
  \setlength{\fboxsep}{2pt}%
  \fbox{\sffamily\mdseries#1}}
```

```
Début
  { une super boucle }
  Pour i variantDe 0 à 9 Faire
    Si i mod 2 = 0 Alors
      Afficher(i," pair")
    Sinon   { là c'est impair }
      Afficher(i," impair")
    FinSi
  FinPour
Fin
```

Notez enfin que vous pouvez utiliser les commandes :

1. À la condition d'inclure le package `ulem` qui définit la commande `\uline` permettant de souligner...

```
\textalgotxt \textalgostr \textalgocom \textalgokw  \textalgotype
```

pour insérer dans le texte une partie d'un algorithme, respectivement : du texte, une chaîne de caractères, un commentaire, un type. Ainsi on pourra utiliser :

```
La déclaration :
\begin{algo}
\TYPE
\DECLTYPE{Ttab}{\ARRAY{entier}}
\ENDTYPE
\end{algo}
définit un type tableau
\textalgotype{Ttab}
d'\textalgotype{entier}.
```

La déclaration :

Type
| *Ttab* = **Tableau** *entier* []

définit un type tableau *Ttab* d'*entier*.

3.9 Commentaires

La fonte utilisée pour les commentaires est définie par la commande `\algocommentfont` (cf. § 3.8 page 15). Les caractères de début et fin de commentaires sont générés par les commandes `\algocommentbeginchars` et `\algocommentendchars`. Ces commandes, pouvant être redéfinies, sont définies par défaut à :

```
\newcommand{\algocommentbeginchars}{\{}
\newcommand{\algocommentendchars}{\}}
```

Il y a deux commandes pour mettre en page les commentaires :
- `\formatcomment` pour les commentaires générés par la commande `\COMMENT` ;
- `\formatinlinecomment` pour les commentaires insérés dans les structures de contrôle (par exemple quand on écrit `\IF[bidule]{i=0}`).

La commande `\formatcomment` est définie comme suit :

```
\newcommand{\formatcomment}[1]{%
  \algocommentbeginchars\ % caractère de début de commentaire et un espace
  {\algocommentfont #1}\ % le commentaire lui-même dans la bonne fonte
  \algocommentendchars}   % le caractère de fin de commentaire
```

La commande `\formatinlinecomment` est définie comme suit :

```
\newcommand{\formatinlinecomment}[1]{%
  \quad% un espace
  \algocommentbeginchars{} % caractère de début de commentaire + un espace
   {\algocommentfont #1 } % commentaire dans la bonne fonte + un espace
   \algocommentendchars}   % le caractère de fin de commentaire
```

On notera donc que c'est au programmeur d'utiliser les commandes pour appeler la fonte pour les commentaires et les caractères de début et fin de commentaires. Voici un exemple de modification des commentaires :

```
\renewcommand{\algocommentbeginchars}{/*}
\renewcommand{\algocommentendchars}{*/}
\renewcommand{\formatcomment}[1]{%
  \algocommentbeginchars\ % caractère de début de commentaire
  \hfill                  % on pousse tout à droite
  {\algocommentfont #1}\ % le commentaire lui-même dans la bonne fonte
  \hfill                  % on pousse à gauche
  \algocommentendchars}   % le caractère de fin de commentaire
\renewcommand{\formatinlinecomment}[1]{%
  \hfill                   % on pousse tout à droite
  \algocommentbeginchars{} % caractère de début de commentaire + un espace
  {\algocommentfont #1 } % commentaire dans la bonne fonte + un espace
  \algocommentendchars}   % le caractère de fin de commentaire
```

```
Début
  /*                                  une super boucle                                  */
  Pour i variantDe 0 à 9 Faire
    Si i mod 2 = 0 Alors
      Afficher(i," pair")
    Sinon                                                          /* là c'est impair */
      Afficher(i," impair")
    FinSi
  FinPour
Fin
```

3.10 Déclarations et arguments formels

Le package `algo` propose à ses vaillants utilisateurs la possibilité de personnaliser la façon d'écrire une déclaration où intervient un identificateur et un type, c'est-à-dire lors de :

- la déclaration de variables et de constantes typées ;
- la déclaration des champs d'un enregistrement ;
- la déclaration des arguments formels d'une procédure ou d'une fonction.

Ainsi par défaut, `\VAR\DECLVAR{i}{entier}\ENDVAR` donne :

```
Variable
  i : entier
```

L'option de package `cstyledecls` :

```
\usepackage[cstyledecls]{algo}
```

permet de produire des déclarations et des arguments formels dans le style du langage C. Les commandes affectées sont :

```
\DECLVAR \DECLFIELD \pfarg \pfargin \pfargout \pfarginout \FUNC
```

On obtiendra par exemple un algorithme ressemblant à :

```
\begin{algo}
  \TYPE
  \RECORD{bidule}
  \DECLFIELD{nom}{chaîne}
  \DECLFIELD{x,y}{flottant}
  \ENDRECORD
  \ENDTYPE
  \VAR
  \DECLVAR{G}{flottant}
  \ENDVAR
  \FUNCTION{incr}{%
    \pfarginout{val}{entier}}{vide}
  \BEGIN
  \STATE{val \recoit{} val + 1}
  \END
  \FUNCTION{plusdeux}{%
    \pfargin{v}{entier}}{entier}
  \BEGIN
  \RETURN{v+2}
  \END
\end{algo}
```

```
Type
  bidule = Enregistrement
    chaîne nom
    flottant x,y
  FinEnregistrement
Variable
  flottant G
vide incr(Ⓣ entier val)
Début
  val ← val + 1
Fin
entier plusdeux(Ⓓ entier v)
Début
  Retourner v+2
Fin
```

Dans la mesure où il n'y a aucune raison de vouloir changer le style des déclarations (mode C ou mode Pascal) au cours d'un document—puisqu'en général, on adopte un formalisme et on s'y tient—nous avons pensé qu'il serait nécessaire d'utiliser les fonctions :

– `\algocstyledecls` pour passer en mode C ;
– `\algopascalstyledecls` pour passer en mode Pascal.

De même on pourra passer en argument optionnel à l'environnement `algo` :
– `declsstyle=c` pour créer un algorithme en mode C si ça n'est pas le mode par défaut ;
– `declsstyle=pascal` pour créer un algorithme en mode Pascal si ça n'est pas le mode par défaut.

Pour détailler quelque peu le fonctionnement des modes « C » et « Pascal » du package `algo`, il faut savoir que l'allure des déclarations est gérée par la commande `\algoformatidtype` définie par défaut comme suit :

```
\newcommand{\algoformatidtype}[2]{#1:{\algotypefont#2}}
```

on écrit donc l'identificateur, suivi du caractère ' :', suivi du type. Pour obtenir une déclaration dans le style du langage C, le package `algo` fait entre autre appel à :

```
\renewcommand{\algoformatidtype}[2]{{\algotypefont#2} #1}
```

qui donnera le type suivi d'un espace, suivi de l'identificateur. Certains pourront donc, s'ils ne sont pas satisfaits des deux modes proposés, créer leur propre style, par exemple avec :

```
\renewcommand{\algoformatidtype}[2]{%
   #1 {\algotypefont(#2)}}
\begin{algo}
  \TYPE
  \RECORD{truc}
  \DECLFIELD{x,y}{flottant}
  \ENDRECORD
  \ENDTYPE
  \VAR
  \DECLVAR{t}{truc}
  \DECLVAR{i}{entier}
  \ENDVAR
\end{algo}
```

Type
  truc = **Enregistrement**
    x,y *(flottant)*
  FinEnregistrement
Variable
  t *(truc)*
  i *(entier)*

4 Algorithmes flottants

On peut vouloir inclure l'environnement algorithme dans un environnement flottant (comme l'environnement `figure`). Pour ce faire a été défini l'environnement `floatalgo` à l'aide du package `float`. Ainsi l'algorithme 1 page suivante a été produit par le code :

```
\begin{floatalgo}[t]
  \begin{algo}[noframe,numbered,noendkeyword]
    \VAR
    \DECLVAR{i,j}{entier}
    \DECLVAR{I}{image}
    \ENDVAR
    \BEGIN
    \FOR{i}{0}{hauteur-1}
    [...]
    \ENDFOR
    \END
  \end{algo}
  \caption{seuillage trivial}
  \label{algo-seuillage}
\end{floatalgo}
```

Comme dans l'exemple de l'algorithme 1 donné ici, on pourra faire référence au numéro grâce au mécanisme de référencement de LaTeX (commande `\ref` et `\label`). Cette fonctionnalité a été définie dans le package `algo` à l'aide des commandes (cf. documentation du package `float`) :

```
\RequirePackage{float}
\floatstyle{ruled}
\newfloat{floatalgo}{thpb}{loa}
\floatname{floatalgo}{Algorithme}
```

Algorithme 1 seuillage trivial

```
1  Variable
2  |  i,j : entier
3  |  I : image
4  Début
5  |  Pour i variantDe 0 à hauteur-1 Faire  { boucle sur la hauteur }
6  |  |  Pour j variantDe 0 à largeur-1 Faire  { boucle sur la largeur }
7  |  |  |  Si I[j][j]>128 Alors
8  |  |  |  |  I[i][j] ← 255  { pixel (i,j) devient blanc }
9  |  |  |  Sinon
10 |  |  |  |  I[i][j] ← 0  { pixel (i,j) devient noir }
11 Fin
```

On constatera donc qu'il est assez aisé de définir son propre environnement flottant si celui-ci ne convenait pas...

5 Retour

J'apprécierais d'avoir un retour d'information sur l'utilisation de ce package qui a été testé sur feu la distribution teTEX, sur la distribution TEXLive installées sur des systèmes Debian et Ubuntu.