

## Comment écrire un algorithme en pseudo-code ?

Ce document donne quelques conseils sur la façon formelle d'écrire un algorithme dans le cours " Information, Calcul et Communication ". Il se focalise donc sur le style, la *syntaxe*.

Le tout premier conseil est justement de **ne pas** commencer par la syntaxe (" *comment* écrire ? ") mais, vraiment, de commencer par le fond/le but (" *quoi* écrire ? ") : ne vous bloquez pas sur comment écrire votre algorithme si vous ne savez pas encore clairement ce que vous voulez écrire. Le premier conseil est donc de réfléchir, faire un/des brouillon(s), schémas, etc.

Une fois au clair sur le " quoi ", et seulement à ce moment-là, préoccupez-vous de la mise en forme. Commencez pour cela par écrire formellement le problème (en français tout de même) par la description la plus précise possible des entrées fournies à l'algorithme et la sortie obtenue.

Par exemple, pour l'algorithme de recherche d'une des valeurs minimales dans une liste, on écrit :

<b>Valeur minimale</b>
entrée : <i>liste L (non-vide) de nombres entiers</i>
sortie : <i>la (ou une des) valeur(s) minimale(s) de la liste</i>
<i>(instructions)</i>

Utilisez ensuite les instructions suivantes :

- affectation :  $\leftarrow$   
p.ex. :  $x \leftarrow 3$
- toutes les opérations mathématiques : notation usuelle  
p.ex. :  $x \geq 2$
- désignation d'un élément d'une liste : parenthèses rondes () ou carrées [], au choix  
p.ex. : le  $i$ -ème élément de la liste  $L$  :  $L(i)$  ou  $L[i]$
- désignation d'un sous-ensemble d'éléments d'une liste : à nouveau, avec des parenthèses rondes ou carrées, au choix, et la notation " $a : b$ " pour désigner un intervalle  
p.ex. :  $L(1 : m)$  désigne les  $m$  premiers éléments de la liste  $L$  (qui peut en contenir plus que  $m$ ).
- les trois structures de contrôle :
  - les tests :  
**Si** condition  
| instructions  
**Sinon**  
| instructions
  - les boucles simples :  
**Pour**  $i$  allant de 1 à  $n$   
| instructions
  - les boucles conditionnelles :  
**Tant que** condition  
| instructions

Veillez noter que dans la boucle simple ci-dessus (celle commençant par **Pour**), la valeur de  $i$  est automatiquement incrémentée de 1 à chaque exécution du coeur de la boucle, tandis que dans une boucle conditionnelle (celle commençant par **Tant que**), c'est à vous de changer quelque chose dans le coeur de la boucle pour que l'algorithme en sorte une fois. Par exemple, si vous écrivez :

```

i ← 1
Tant que i ≤ n
|   Afficher : i

```

L'algorithme affichera une suite infinie de 1 sans jamais s'arrêter. Pour éviter ça, il faut manuellement rajouter l'instruction  $i \leftarrow i + 1$  dans le coeur de la boucle.

- Voici deux autres exemples de boucles simples avec une autre incrémentation :

```

Pour i allant de 1 à n de 2 en 2
|   instructions

```

et

```

Pour i allant de n à 1 en descendant
|   instructions

```

- si l'ensemble décrit par la boucle est l'ensemble vide, la boucle ne se déroule pas du tout ; p.ex.

**Pour** i allant de 1 à n  
ne fera *rien* si n est inférieur ou égal à 0.

**Remarque :** Pensez à indenter (décaler à droite) et même marquer par une barre verticale, les instructions contrôlées par une structure de contrôle.

- La terminaison de l'algorithme : “ **Sortir** : ” ;

p.ex. : **Sortir** : x ;

notez que l'instruction “ **Sortir** : ” met fin à l'algorithme (même s'il y a encore des lignes en dessous) ;

- si nécessaire (rare dans des algorithmes formels), pour afficher une valeur/expression, utilisez simplement “ **Afficher** : ” ;

p.ex. : **Afficher** : x.

Sauf mention contraire dans la donnée, vous pouvez également utiliser tout algorithme *vu en cours* (taille, tri, recherche, plus court chemin) en le désignant par un nom suffisamment clair ; par exemple :

- $n \leftarrow \mathbf{taille}(L)$

- $L' \leftarrow \mathbf{tri\ par\ insertion}(L)$

*Note :* Au niveau formel, il est préférable de considérer que les algorithmes ne modifient pas leur entrée mais produisent un nouvel objet (comme une fonction mathématique). Par exemple, ci-dessus, la liste  $L$  n'est pas modifiée par l'algorithme de tri, mais celui-ci retourne une nouvelle liste (triée).

Terminons par un exemple complet : l'algorithme de recherche d'une des valeurs minimales dans une liste :

Valeur minimale
entrée : liste $L$ ( <i>non-vide</i> ) de nombres
sortie : la (ou une des) valeur(s) minimale(s) de la liste
<pre> n ← <b>taille</b>(L) x<sub>min</sub> ← L(1) <b>Pour</b> i allant de 2 à n     <b>Si</b> L(i) &lt; x<sub>min</sub>         x<sub>min</sub> ← L(i) <b>Sortir</b> : x<sub>min</sub> </pre>

Notez que l'algorithme ci-dessus est correct dans tous les cas en raison des conventions :

- la boucle simple “**Pour ...**” ne fait rien si  $n$  vaut 1 (et donc, dans ce cas, on retourne finalement  $L(1)$ ) ;
- la description de l'entrée est toujours vraie : ci-dessus la liste  $L$  ne peut (axiomatiquement) pas être vide ; il est donc important de bien préciser les hypothèses de départ.