Comment écrire proprement un algorithme?

Jean-Cédric Chappelier

Version 1.2 – nov. 2025

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 maximales dans une liste, on écrit :

Valeur maximale
entrée: L *une liste non vide de nombres* sortie: *la (ou une des) valeur(s) maximale(s) de la liste*

Utilisez ensuite les instructions suivantes :

— affectation : $\longleftarrow$
p.ex. : $x \longleftarrow 3$

— 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 $L[i]$

- les trois structures de contrôle :
  - branchements conditionnels :

    Si condition
    | instructions

    Sinon
    | instructions

  - boucles conditionnelles :

    Tant que condition
    | instructions

    Répéter
    | instructions
    tant que condition

  - itérations :

    Pour tout élément x de L
    | instructions

    ou

    Pour i allant de 1 à n
    | instructions

    Les conventions d'écriture des boucles « Pour tout » incluent :
    - que sur les nombres entiers l'incrément est de 1; sinon il faut le préciser; p.ex. :

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

      autre exemple :

      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.

  - i et n (ou L dans le premier cas) ne doivent pas être modifiés dans la boucle, sinon le comportement n'est pas défini. Utiliser une boucle conditionnelle dans de tels cas.

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 :

— $n \longleftarrow$ **taille**(L)

— $L' \longleftarrow$ **trier**(L) ou $L' \longleftarrow$ **tri**(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).

Donnons un exemple complet : un algorithme de recherche d'*une* des valeurs maximales dans une liste :

Valeur maximale

entrée: L *une liste non vide de nombres*
sortie: *la valeur maximale de la liste*

$n \longleftarrow$ ***taille***(L)
$x_{\max} \longleftarrow L[1]$
Pour i *allant de* 2 *à* n
    Si $L[i] > x_{\max}$
        $x_{\max} \longleftarrow L[i]$
Sortir : $x_{\max}$

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

— la boucle « Pour tout » 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. Par exemple l'algorithme suivant n'est pas correct :

Valeur maximale
entrée: *L une liste de nombres* sortie: *la (ou une des) valeur(s) maximale(s) de la liste*
$n \longleftarrow$ ***taille***(L) $x_{\max} \longleftarrow L[1]$ *etc.*

car $L[1]$ n'est pas défini pour une liste vide (et que l'on n'a pas empêché cette possibilité *a priori*). Il faut donc l'écrire comme donné plus haut, et pas autrement, car il n'y a de toutes façons pas de définition de « la valeur maximale » pour une liste vide.

Donnons maintenant quelques notations usuelles d'opérations sur des listes.

Pour ajouter des éléments à une liste, ou aussi concaténer (fusionner l'une derrière l'autre) deux listes, vous pouvez, si vous le souhaitez, utiliser la notation $\oplus$: $L_1 \oplus L_2$ crée la liste de tous les éléments de L_1 suivis de tous les éléments de L_2. Notez que c'est une opération sur des *listes*.

Pour un élément à une liste, on peut alors simplement écrire :
pour un ajout en fin : $L \longleftarrow L \oplus (x)$
pour un ajout au début : $L \longleftarrow (x) \oplus L$

Comment ajouter un élément x à une position i dans une liste ?
Le plus simple est de l'écrire explicitement (la taille de L ayant au préalable été affectée à n) :

$$L \longleftarrow \big(L[1], ..., L[i-1], x, L[i], ..., L[n]\big)$$

avec la convention que si i vaut 1, alors x est ajouté au début (mais préférez si possible la notation ci-dessus avec $\oplus$) et que si i vaut $n+1$, x est ajouté en fin (mais, là aussi, préférez si possible la notation ci-dessus avec $\oplus$).

Comment vider une liste ?
Il suffit simplement de lui affecter la liste vide : $L \longleftarrow ()$

Comment supprimer un élément d'une liste ?
En écrivant explicitement la liste sans cet élément :

$$L \longleftarrow (L[1], ..., L[i-1], L[i+1], ..., L[n])$$

Mais pourquoi ne peut-on pas noter $L \ominus L[i]$? (ne **jamais** écrire ça !)
Parce que cette notation est *ambiguë* : évalué à droite d'une expression (« *r-value* »), l'expression $L[i]$ représente une **valeur**. Par exemple $a \longleftarrow L[i]$ veut bien dire $a \longleftarrow 4$ si $L[i]$ vaut 4.

Que voudrait alors dire $L \ominus 4$ si L contient plusieurs fois la valeur 4 ? Lequel supprime-t-on ? Ou les supprime-t-on tous ?

On pourrait à la limite tolérer une notation non ambiguë telle que $L \ominus i$ (où i désigne alors une position/un indice), mais je pense que ce ne serait pas très bon pédagogiquement (encore une notation de plus ; risque de confusion avec la notation critiquée ci-dessus ; ...).

Terminons ce document sur l'écriture d'algorithmes par une autre erreur fréquente :
il ne faut pas écrire de condition comme « $a = b = c$» ; et cela pour deux raisons :

— premièrement en raison du « sinon » : que veut dire le « sinon » ?
Est-ce
« $(a \neq b)$ **OU** $(b \neq c)$ »
(b est différent de l'un ou l'autre), ou alors est-ce
« $(a \neq b)$ **ET** $(b \neq c)$ **ET** $(a \neq c)$ »
(les trois sont différents) ?

Autrement dit, est-ce « $a = b = c$ » signifie
« $(a = b)$ **ET** $(b = c)$ »
ou
« $(a = b)$ **OU** $(b = c)$ **OU** $(a = c)$ »
?

Il y a en fait quatre situations différentes possibles pour la « négation de » xxx : xxy, xyx, yxx et xyz...
Lesquelles veut-on (toutes ou seulement certaines) ?

— de plus, si l'on autorise « $a = b = c$ », alors c'est la porte ouverte à « $a \neq b \neq c$ »...
...qui est clairement ambiguë (quid de la relation entre a et c ?).