Table des matières

1 Équivalence entre « pseudo-codes »

Résumé des différences entre le pseudo-code utilisé par AlgoBox et celui que l'on peut rencontrer dans les manuels.

1.1 Entrée des données

Pseudo-code AlgoBox	Variantes
`LIRE...`	`Saisir...` `Entrer...`

1.2 Affichage des données

Pseudo-code AlgoBox	Variantes
`AFFICHER...`	`Écrire...`

1.3 Affecter une valeur à une variable

<table>
<tr><th>Pseudo-code AlgoBox</th><th>Variantes</th></tr>
<tr><td><pre>...PREND_LA_VALEUR...</pre></td><td><pre>Affecter...à...</pre></td></tr>
</table>

1.4 Structure SI ALORS

<table>
<tr><th>Pseudo-code AlgoBox</th><th>Variantes</th></tr>
<tr><td><pre>SI...ALORS
  DEBUT_SI
  ...
  FIN_SI</pre></td><td><pre>Si...alors
  ...
FinSi</pre>
<pre>Si...
Alors...
FinSi</pre></td></tr>
</table>

<table>
<tr><th>Pseudo-code AlgoBox</th><th>Variantes</th></tr>
<tr><td><pre>SI...ALORS
  DEBUT_SI
  ...
  FIN_SI
  SINON
    DEBUT_SINON
    ...
    FIN_SINON</pre></td><td><pre>Si...alors
  ...
  sinon
  ...
FinSi</pre>
<pre>Si...
Alors...
Sinon...
FinSi</pre></td></tr>
</table>

1.5 Boucle POUR...

<table>
<tr><th>Pseudo-code AlgoBox</th><th>Variantes</th></tr>
<tr><td><pre>POUR...ALLANT_DE...A...
  DEBUT_POUR
  ...
  FIN_POUR</pre></td><td><pre>Pour...de...jusqu'à...
  ...
FinPour</pre>
<pre>Pour...variant de...à...
  ...
FinPour</pre></td></tr>
</table>

1.6 Structure TANT QUE...

<table>
<tr><th>Pseudo-code AlgoBox</th><th>Variantes</th></tr>
<tr><td><pre>TANT_QUE...FAIRE
  DEBUT_TANT_QUE
  ...
  FIN_TANT_QUE</pre></td><td><pre>Tant que...
  ...
FinTantQue</pre></td></tr>
</table>

2 Les problèmes de syntaxe

2.1 Les erreurs de syntaxe les plus courantes

- *Erreur classique n°1 :* utiliser la virgule au lieu du point comme séparateur décimal
- *Erreur classique n°2 :* utiliser la syntaxe "`SI x=y`" au lieu de "`SI x==y`" pour vérifier une égalité dans une condition
- *Erreur classique n°3 :* utiliser la syntaxe `x^y` au lieu de `pow(x,y)` pour les puissances.

2.2 Syntaxe des opérations mathématiques

Opération mathématique	syntaxe AlgoBox
$\sqrt{x}$	`sqrt(x)`
x^y	`pow(x,y)`
$\cos(x)$ (x en radians)	`cos(x)`
$\sin(x)$ (x en radians)	`sin(x)`
$\tan(x)$ (x en radians)	`tan(x)`
e^x	`exp(x)`
$\ln x$	`log(x)`
$\|x\|$	`abs(x)`
Partie entière de x	`floor(x)`
Nombre pseudo-aléatoire compris entre 0 et 1	`random()`
Reste de la division euclidienne de n par p	`n%p`
π	`Math.PI`
$\arccos(x)$	`acos(x)`
$\arcsin(x)$	`asin(x)`
$\arctan(x)$	`atan(x)`

2.3 Syntaxe pour les conditions

Condition logique	syntaxe AlgoBox
$x = y$	`x==y`
$x \neq y$	`x!=y`
$x < y$	`x<y`
$x \leqslant y$	`x<=y`
$x > y$	`x>y`
$x \geqslant y$	`x>=y`
condition1 **ou** condition2	`condition1 OU condition2`
condition1 **et** condition2	`condition1 ET condition2`

Rappel : il est possible d'inclure des opérations mathématiques dans les conditions (exemple : `pow(2,n)<100`)

2.4 Syntaxe pour les fonctions statistiques et les opérations sur les listes

Somme des termes d'une liste	`ALGOBOX_SOMME(nom_de_la_liste,rang_premier_terme,rang_dernier_terme)`
Moyenne	`ALGOBOX_MOYENNE(nom_de_la_liste,rang_premier_terme,rang_dernier_terme)`
Variance	`ALGOBOX_VARIANCE(nom_de_la_liste,rang_premier_terme,rang_dernier_terme)`
Écart-type	`ALGOBOX_ECART_TYPE(nom_de_la_liste,rang_premier_terme,rang_dernier_terme)`
Médiane	`ALGOBOX_MEDIANE(nom_de_la_liste,rang_premier_terme,rang_dernier_terme)`
Q1 (version calculatrice)	`ALGOBOX_QUARTILE1(nom_de_la_liste,rang_premier_terme,rang_dernier_terme)`
Q3 (version calculatrice)	`ALGOBOX_QUARTILE3(nom_de_la_liste,rang_premier_terme,rang_dernier_terme)`
Q1 (version « officielle »)	`ALGOBOX_QUARTILE1_BIS(nom_de_la_liste,rang_premier_terme,rang_dernier_terme)`
Q3 (version « officielle »)	`ALGOBOX_QUARTILE3_BIS(nom_de_la_liste,rang_premier_terme,rang_dernier_terme)`
Minimum d'une liste	`ALGOBOX_MINIMUM(nom_de_la_liste,rang_premier_terme,rang_dernier_terme)`
Maximum d'une liste	`ALGOBOX_MAXIMUM(nom_de_la_liste,rang_premier_terme,rang_dernier_terme)`
Rang du minimum	`ALGOBOX_POS_MINIMUM(nom_de_la_liste,rang_premier_terme,rang_dernier_terme)`
Rang du maximum	`ALGOBOX_POS_MAXIMUM(nom_de_la_liste,rang_premier_terme,rang_dernier_terme)`

2.5 Autres fonctions

entier pseudo-aléatoire entre p et n	`ALGOBOX_ALEA_ENT(p,n)`
$\binom{n}{p}$ $(*)$	`ALGOBOX_COEFF_BINOMIAL(n,p)`
$p(X=k)$ pour la loi binomiale $(*)$	`ALGOBOX_LOI_BINOMIALE(n,p,k)`
$p(X<x)$ pour la loi normale centrée réduite	`ALGOBOX_LOI_NORMALE_CR(x)`
$p(X<x)$ pour la loi normale	`ALGOBOX_LOI_NORMALE(esp,ecart,x)`
x tel $p(X<x)=p$ pour la loi normale centrée réduite	`ALGOBOX_INVERSE_LOI_NORMALE_CR(p)`
x tel $p(X<x)=p$ pour la loi normale	`ALGOBOX_INVERSE_LOI_NORMALE(esp,ecart,p)`
$n!$ $(*)$	`ALGOBOX_FACTORIELLE(n)`

$(*)$: avec $n < 70$

3 Fonctionnement d'AlgoBox

3.1 Les deux règles fondamentales

1. Toute variable doit d'abord être déclarée avant de pouvoir être utilisée. La première chose à faire avant de concevoir un algorithme est d'ailleurs de lister toutes les variables qui seront nécessaires.
2. Une nouvelle instruction ne peut s'insérer que sur une ligne vierge.

3.2 Les variables

Attention

Le nom des variables ne doit pas contenir d'espaces (que l'on peut remplacer par le caractère `_`), ni d'accents, ni de caractères spéciaux. On ne peut pas non plus utiliser certains termes réservés : par exemple, une variable ne peut pas être appelée `NOMBRE`. Il est par contre conseillé d'utiliser des noms de variables explicites (mêmes longs) pour rendre les algorithmes plus clairs.

3.3 Les listes de nombres

Il est possible d'entrer directement les termes d'une liste. Pour cela, il suffit d'utiliser l'instruction `LIRE maliste[1]` (1 représentant le premier rang des termes de la liste que l'on veut entrer). Lors de l'exécution de l'algorithme, il suffira alors d'entrer toutes les valeurs souhaitées (dans l'ordre) en les séparant par `:` .

3.4 Boucle POUR...DE...A

- On n'utilise ce genre de boucles que si on connaît à l'avance le nombre de répétitions à effectuer.
- La variable servant de compteur pour la boucle doit être du type NOMBRE et doit être déclarée préalablement (comme toutes les variables).
- Dans AlgoBox, cette variable est automatiquement augmentée de 1 à chaque fois.

Attention

- On peut utiliser la valeur du compteur pour faire des calculs à l'intérieur de la boucle, mais les instructions comprises entre `DEBUT_POUR` et `FIN_POUR` ne doivent en aucun cas modifier la valeur de la variable qui sert de compteur.
- Le nombre d'itérations sur AlgoBox est limité à 500 000. En cas de dépassement, l'algorithme s'arrête et le message « ***Algorithme interrompu suite à une erreur*** » est affiché.
- Si les instructions à répéter comportent l'affichage d'une variable ou un tracé graphique, il faut limiter le nombre d'itérations à moins d'un millier (ou guère plus) : sans quoi, l'exécution du programme risque de prendre beaucoup trop de temps. Par contre, s'il n'y que des calculs à répéter on peut pousser le nombre d'itérations plus loin.

3.5 Structure TANT QUE

- Cette structure sert à répéter des instructions que l'on connaisse ou non par avance le nombre d'itérations à effectuer.
- Si la condition du TANT QUE... est fausse dès le début, les instructions entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE` ne sont jamais exécutées (la structure TANT QUE ne sert alors strictement à rien).

Attention

- Il est indispensable de s'assurer que la condition du TANT QUE... finisse par être vérifiée (le code entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE` doit rendre vraie la condition tôt ou tard), sans quoi l'algorithme va se lancer dans une « boucle infinie ».
- Afin d'éviter les boucles infinies, le nombre d'itérations est là aussi limité à 500 000. En cas de dépassement, l'algorithme s'arrête et le message « ***Algorithme interrompu suite à une erreur*** » est affiché.
- Si les instructions à répéter comportent l'affichage d'une variable ou un tracé graphique, il faut limiter le nombre d'itérations à moins d'un millier (ou guère plus) : sans quoi, l'exécution du programme risque de prendre beaucoup trop de temps. Par contre, s'il n'y que des calculs à répéter on peut pousser le nombre d'itérations plus loin.

3.6 Utilisation de l'onglet « Utiliser une fonction numérique »

En activant l'option *"Utiliser une fonction"* dans l'onglet *"Utiliser une fonction numérique"*, on peut utiliser l'image de n'importe quel nombre (ou variable de type nombre) par la fonction notée `F1` dans le code de l'algorithme. Il suffit pour cela d'entrer l'expression de `F1(x)` en fonction de `x` dans le champ prévu pour cela.
Pour utiliser l'image d'un nombre `nb` par la fonction `F1` dans l'algorithme, il suffit d'utiliser le code : `F1(nb)` (cela peut se faire dans une affectation ou dans une expression conditionnelle).
Cette option est particulièrement utile quand un algorithme nécessite le calcul de plusieurs images par une même fonction.
Un exemple classique est celui de la dichotomie :

```
VARIABLES
  precision EST_DU_TYPE NOMBRE
  a EST_DU_TYPE NOMBRE
  b EST_DU_TYPE NOMBRE
  m EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  a PREND_LA_VALEUR ...
  b PREND_LA_VALEUR ...
  LIRE precision
  TANT_QUE (b-a>precision) FAIRE
    DEBUT_TANT_QUE
    m PREND_LA_VALEUR (a+b)/2
    SI (F1(m)*F1(b)>0) ALORS
      DEBUT_SI
      b PREND_LA_VALEUR m
      FIN_SI
      SINON
        DEBUT_SINON
        a PREND_LA_VALEUR m
        FIN_SINON
    AFFICHER a
    AFFICHER " < solution < "
    AFFICHER b
    FIN_TANT_QUE
FIN_ALGORITHME
```

3.7 Utilisation de l'onglet « Dessiner dans un repère »

En activant l'option *"Utiliser un repère"* dans l'onglet *"Dessiner dans un repère"*, un repère graphique est automatiquement ajouté dans la fenêtre de test de l'algorithme. Il est alors possible d'inclure dans le code de l'algorithme des instructions pour tracer des points et des segments dans ce repère en utilisant les boutons *"Ajouter TRACER POINT"* et *"Ajouter TRACER SEGMENT"*.

Attention

La « fenêtre » du repère est définie lors de la première utilisation des instructions `TRACER_POINT` et `TRACER_SEGMENT`. Si la fenêtre doit dépendre de la valeur de certaines variables, il faut s'assurer que celles-ci sont bien définies avant le premier tracé.

Exemple : représentation graphique d'une fluctuation d'échantillonnage (100 simulations de 1000 lancers d'une pièce)

```
VARIABLES
  simulation EST_DU_TYPE NOMBRE
  lancer EST_DU_TYPE NOMBRE
  nb_de_piles EST_DU_TYPE NOMBRE
  frequence EST_DU_TYPE LISTE
  moy EST_DU_TYPE NOMBRE
  ecart_type EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  POUR simulation ALLANT_DE 1 A 100
    DEBUT_POUR
    nb_de_piles PREND_LA_VALEUR 0
    POUR lancer ALLANT_DE 1 A 1000
      DEBUT_POUR
      SI (random()<0.5) ALORS
        DEBUT_SI
        nb_de_piles PREND_LA_VALEUR nb_de_piles+1
        FIN_SI
      FIN_POUR
    frequence[simulation] PREND_LA_VALEUR nb_de_piles/1000
    TRACER_POINT (simulation,frequence[simulation])
    FIN_POUR
  moy PREND_LA_VALEUR ALGOBOX_MOYENNE(frequence,1,100)
  TRACER_SEGMENT (1,moy)->(100,moy)
  ecart_type PREND_LA_VALEUR ALGOBOX_ECART_TYPE(frequence,1,100)
  TRACER_SEGMENT (1,moy+2*ecart_type)->(100,moy+2*ecart_type)
  TRACER_SEGMENT (1,moy-2*ecart_type)->(100,moy-2*ecart_type)
FIN_ALGORITHME

Xmin: 1 ; Xmax: 100 ; Ymin: 0.4 ; Ymax: 0.6 ; GradX: 10 ; GradY: 0.01
```

3.8 Utilisation de l'onglet « Fonction avancée »

En activant l'option *"Utiliser la fonction F2..."* dans l'onglet *"Fonction avancée"*, on peut définir :
- une fonction définie par « morceaux » ;
- une fonction dépendant de plusieurs paramètres (contrairement à la fonction F1 qui ne doit dépendre que de x) ;
- une fonction définie de façon récursive.

Pour cela, il faut :
- préciser les paramètres dont dépend la fonction ;
- ajouter une à une les conditions permettant de définir la fonction ;
- indiquer ce que doit retourner la fonction quand aucune des conditions n'est remplie.

Exemple avec le calcul du pgcd de deux entiers (méthode récursive) :

```
VARIABLES
  p EST_DU_TYPE NOMBRE
  q EST_DU_TYPE NOMBRE
  pgcd EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  LIRE p
  LIRE q
  pgcd PREND_LA_VALEUR F2(p,q)
  AFFICHER pgcd
FIN_ALGORITHME

fonction F2(p,q):
SI (p==0) RENVOYER q
SI (q==0) RENVOYER p
SI (q<=p) RENVOYER F2(p-q,q)
Dans les autres cas, RENVOYER F2(p,q-p)
```

3.9 Récupération facile d'un code AlgoBox dans un traitement de texte

Pour copier un code AlgoBox facilement sans passer par les commandes d'exportation du menu *Fichier*, il suffit de :
- lancer la fenêtre de test de l'algorithme ;
- sélectionner le code à copier dans la page web de test (la partie supérieure de la fenêtre) ;
- faire « glisser » le code sélectionné vers son traitement de texte.

4 Quelques techniques classiques

4.1 Diviseur ?

- Condition pour vérifier si un entier P est un diviseur d'un entier N : `N%P==0`

4.2 Entier pair ou impair ?

- Condition pour vérifier si un entier N est pair : `N%2==0`
- Condition pour vérifier si un entier N est impair : `N%2!=0` (autre condition possible : `N%2==1`)

4.3 Entier pseudo-aléatoire compris entre 1 et N

- Pour obtenir un entier pseudo-aléatoire compris entre 1 et N : `... PREND_LA_VALEUR ALGOBOX_ALEA_ENT(1,N)` ou `... PREND_LA_VALEUR floor(N*random()+1)`
- Exemple pour la face d'un dé : `... PREND_LA_VALEUR ALGOBOX_ALEA_ENT(1,6)`

4.4 « Balayage » d'un intervalle

Méthodes pour « balayer » un intervalle $[a ; b]$ (a et b faisant partie du traitement) :

En entrant le nombre de subdivisions et en utilisant une boucle POUR... :

```
VARIABLES
  i EST_DU_TYPE NOMBRE
  nbsubdivisions EST_DU_TYPE NOMBRE
  pas EST_DU_TYPE NOMBRE
  x EST_DU_TYPE NOMBRE
  a EST_DU_TYPE NOMBRE
  b EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  LIRE a
  LIRE b
  LIRE nbsubdivisions
  pas PREND_LA_VALEUR (b-a)/nbsubdivisions
  POUR i ALLANT_DE 0 A nbsubdivisions
    DEBUT_POUR
    x PREND_LA_VALEUR a+i*pas
    traitement....
    FIN_POUR
FIN_ALGORITHME
```

En entrant directement le pas et en utilisant une structure TANT QUE... :

```
VARIABLES
  pas EST_DU_TYPE NOMBRE
  x EST_DU_TYPE NOMBRE
  a EST_DU_TYPE NOMBRE
  b EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  LIRE a
  LIRE b
  LIRE pas
  x PREND_LA_VALEUR a
  TANT_QUE (x<=b) FAIRE
    DEBUT_TANT_QUE
    traitement....
    x PREND_LA_VALEUR x+pas
    FIN_TANT_QUE
FIN_ALGORITHME
```

4.5 Suites numériques

D'un strict point de vue programmation, l'utilisation d'une liste pour manipuler les termes d'une suite dans un algorithme n'est guère optimal. Par contre, d'un point de vue pédagogique, la correspondance étroite entre terme d'une suite et terme d'une liste AlgoBox peut permettre de conforter la compréhension par les élèves du formalisme sur les suites numériques.

Pour modéliser une suite (U_n) à l'aide d'un algorithme AlgoBox, on peut utiliser une variable de type `LISTE` notée `U`. Ainsi :

- le terme U_0 de la suite sera représenté par `U[0]` dans l'algorithme ;
- le terme U_1 de la suite sera représenté par `U[1]` dans l'algorithme ;
- etc...
- le terme U_n de la suite sera représenté par `U[n]` dans l'algorithme ;
- le terme U_{n+1} de la suite sera représenté par `U[n+1]` dans l'algorithme ;

Exemple avec une suite « récurrente » :

Sans utiliser de liste :

```
VARIABLES
  n EST_DU_TYPE NOMBRE
  U EST_DU_TYPE NOMBRE
  i EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  LIRE n
  U PREND_LA_VALEUR 1
  POUR i ALLANT_DE 0 A n-1
    DEBUT_POUR
    U PREND_LA_VALEUR 1+2*U
    FIN_POUR
  AFFICHER U
FIN_ALGORITHME
```

En utilisant une liste AlgoBox :

```
VARIABLES
  n EST_DU_TYPE NOMBRE
  U EST_DU_TYPE LISTE
  i EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  LIRE n
  U[0] PREND_LA_VALEUR 1
  POUR i ALLANT_DE 0 A n-1
    DEBUT_POUR
    U[i+1] PREND_LA_VALEUR 1+2*U[i]
    FIN_POUR
  AFFICHER U[n]
FIN_ALGORITHME
```

4.6 Échanger le contenu de deux variables

Pour échanger le contenu de deux variables `A` et `B`, il suffit d'utiliser une troisième variable `temp` :

```
temp PREND_LA_VALEUR A
A PREND_LA_VALEUR B
B PREND_LA_VALEUR temp
```

Exemple : tri d'une liste de valeurs par échange du minimum

```
VARIABLES
  listenombre EST_DU_TYPE LISTE
  temp EST_DU_TYPE NOMBRE
  i EST_DU_TYPE NOMBRE
  min EST_DU_TYPE NOMBRE
  nbtermes EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  LIRE nbtermes
  //Entrer les "nbtermes" valeurs sous la forme valeur1:valeur2:etc...
  LIRE listenombre[1]
  POUR i ALLANT_DE 1 A nbtermes-1
    DEBUT_POUR
    min PREND_LA_VALEUR ALGOBOX_POS_MINIMUM(listenombre,i+1,nbtermes)
    SI (listenombre[i]>listenombre[min]) ALORS
      DEBUT_SI
      temp PREND_LA_VALEUR listenombre[min]
```

```
      listenombre[min] PREND_LA_VALEUR listenombre[i]
      listenombre[i] PREND_LA_VALEUR temp
      FIN_SI
    FIN_POUR
  POUR i ALLANT_DE 1 A nbtermes
    DEBUT_POUR
    AFFICHER listenombre[i]
    AFFICHER " "
    FIN_POUR
FIN_ALGORITHME
```

5 Bogues connus

« `continue` » ne doit pas être utilisé comme nom de variable (il devrait être interdit par AlgoBox, ce qui n'est pas le cas dans la version actuelle)