Cours d'algorithmique pour la classe de 2nde

F.Gaudon

30 août 2012

Table des matières

1 Avant la programmation

1.1 Qu'est ce qu'un algorithme ?

Définition :

> Un *algorithme* est une succession d'*instructions* (aussi appelées *commandes*) permettant la résolution d'un problème donné.

Remarque :
Le terme « algorithme » vient du nom du mathématicien arabe du IX^{e} siècle *Al Khuwarizmi* qui écrivit une des premières méthodes systématiques de résolution de certaines équations.

Exemple :

```
Pour a allant de 1 à 10 par pas de 1 ;
stocker dans b le carré de a ;
afficher b.
```

L'algorithme précédent calcule et affiche le carré des nombres de 1 à 10. Dans cet algorithme,`Afficher` b est une instruction.

1.2 Qu'est ce qu'un langage de programmation ?

Définition :

> Un *langage de programmation* est un ensemble d'instructions et de règles syntaxiques compréhensible par l'ordinateur et permettant de créer des algorithmes. Un *programme* est la traduction d'un algorithme dans le langage de programmation utilisé.

Exemples :
BASIC, PASCAL, C++, assembleur sont des langages de programmation pour ordinateurs. Dans ce cours nous utiliserons les langages de programmation associés aux calculatrices programmables Casio et Texas Instrument ainsi que le langage de programmation du logiciel libre et gratuit XCas téléchargeable à l'adresse `www-fourier.ujf-grenoble.fr/~parisse/giac_fr.html` et le langage de programmation du logiciel libre et gratuit MAXIMA téléchargeable à l'adresse suivante `http://michel.gosse.free.fr` (on installera l'interface graphique WxMaxima qui simplifie beaucoup l'utilisation de ce logiciel) .Ces deux logiciels sont par ailleurs des puissants logiciels de calcul dit formel utilisés dans le monde universitaire, mais la description de leurs possibilités n'est pas l'objet de ce cours.

1.3 Avant de programmer

1.3.1 Créer ou modifier ou exécuter un programme

Casio :
Touche [MENU] puis choisir [PRGM] et :
- [EDIT] pour modifier un programme existant ;
- [NEW] pour créer un nouveau programme ;
- [EXEC] pour exécuter un programme.

TI :
Touche [PRGM] puis :
- [EDIT] pour modifier un programme existant ;
- [NEW] pour créer un nouveau programme ;
- [EXEC] pour exécuter un programme existant.

Remarque :
Après création d'un nouveau programme sur TI ou CASIO, entrer le nom du programme ; n'utiliser que les lettres (touches [ALPHA] + Lettre)

XCas :
L'édition d'un programme se fait dans une fenêtre de programmation que l'on ouvre en tapant simultanément [ALT] et P. Avant de commencer, aller dans le menu [Cfg] [configuration du CAS] et vérifier que l'onglet [PROG STYLE] est en mode XCAS. On pourra aussi aller dans [Cfg] [Polices (Toutes)] et choisir une police de taille 14 plus lisible que la police de taille 18 par défaut.

Python :
L'édition d'un programme se fait dans un logiciel d'édition de texte (notepad, wordpad, gedit, geany,...)

1.3.2 Instructions d'un programme

Casio :
Les instructions des algorithmes peuvent être séparées par un retour à la ligne [EXE]. Une ligne peut éventuellement comporter plusieurs instructions séparées par [:].

TI :
Les instructions des algorithmes peuvent être séparées par un retour à la ligne [EXE]. Une ligne peut éventuellement comporter plusieurs instructions séparées par [:].

XCas :
Les instructions ou les blocs d'instructions (suite d'instructions liées dont on verra des exemples dans ce cours) doivent être séparés par un point virgule [;] et un retour à la ligne [SHIFT] [ENTER]. Une ligne peut contenir plusieurs instructions séparées par [;].

Python :
Les instructions doivent être séparées par un retour à la ligne.

2 Les variables

Définition :

On appelle variable tout emplacement de la mémoire de l'ordinateur ou de la calculatrice dans lequel on stocke une information qui peut être changée. Une variable est donc constituée :
- d'un nom qui permet de reconnaître où elle se situe dans la mémoire de l'ordinateur ou de la calculatrice ;
- d'une valeur : le nombre ou plus généralement l'information stockée.

Remarque :
Les variables sous Casio ou TI peuvent contenir uniquement des nombres. Sous XCas, Maxima et autres langages de programmation pour ordinateur, les variables peuvent contenir des caractères, des lettres, des chaînes de caractères.

Syntaxe :
Sur Casio ou TI, on écrira 3 $\boxed{\rightarrow}$ A pour stocker le nombre 3 dans la variable A. Sur TI, la touche correspondante est $\boxed{\text{STO}\blacktriangleright}$ et sur casio $\boxed{\rightarrow}$. Sur XCas, on écrira $a\boxed{:}\boxed{=}3$ et avec Python, on écrira $a\boxed{=}3$ pour stocker la valeur 3 dans la variable a.

Déclaration des variables :

Dans un algorithme, on commence généralement par énoncer les variables qui seront employées et la nature des données que l'on stockera dans la variable : nombres entiers, nombres décimaux, caractères, chaînes de caractères. On dit que l'on « déclare » les variables.

Exemple :
Dans l'algorithme de calcul de carrés précédents, il y a deux variables utilisées a et b dans lesquelles on stockera des nombres réels. L'algorithme peut se reformuler sous la forme plus structurée suivante :

```
Variables a, b
Début traitement
    pour a allant de 1 à 10 faire
        stocker a^2 dans b ;
        afficher b ;
    fin
Fin
```

3 Exercices sur les variables

Exercice 1 :
On considère l'algorithme suivant :

Variables a, b, c : *nombres réels*
Début traitement
  stocker 3 **dans** a ;
  stocker 4 **dans** b ;
  stocker a **dans** c ;
  stocker b **dans** a ;
  stocker c **dans** b
Fin

a) Quel nombre est stocké dans la variable A à l'issue de l'algorithme ? Et dans la variable B ?
b) À quoi sert l'algorithme précédent ?

Exercice 2 :
On considère l'algorithme suivant :

Variables a, b, c : *nombres réels*
Début traitement
  stocker 3 **dans** a ;
  stocker $4 \times a$ **dans** b ;
  stocker $b + 5$ **dans** c ;
Fin

Quel nombre est stocké dans la variable c à la fin de l'algorithme ?

Exercice 3 :
On considère l'algorithme suivant :

Variables x, a, b, c : *nombres réels*
Début traitement
  stocker 2 **dans** x ;
  stocker x^2 **dans** a ;
  stocker $3 \times x$ **dans** b ;
  stocker $a - b + 4$ **dans** c ;
Fin

Quel nombre est stocké dans la variable c à la fin de l'algorithme ?

4 Entrées et sorties

4.1 Commandes d'affichage

Définition :

> Les commandes d'affichage servent à afficher à l'écran du texte ou la valeur d'une variable.

Syntaxe en algorithmique :
Afficher `a` ou Afficher `"texte"`

Casio :

> " `texte` "
> *affiche le texte entre guillemets.*
> A ⊿
> *affiche la valeur de A et attend que l'utilisateur tape sur EXE pour poursuivre l'exécution du programme.*

Les guillemets " sont accessibles à partir de l'éditeur de programmes (MENU) en faisant défiler avec la touche ▶ puis en utilisant la touche Fn qui correspond (F2 pour la Graph25). Le caractère ⊿ est accessible à partir de l'éditeur de programme (MENU PRGM) puis SHIFT PRGM puis ▶ puis en utilisant la touche Fn qui correspond (F2 sur Graph25).

TI :

> Disp " `texte` " , `A`
> *affiche le texte entre guillemets puis le contenu de la variable A.*

La commande Disp ("display" en anglais c'est à dire "afficher") est accessible dans le menu PRGM I/O ("Input/Output" en anglais c'est à dire "entrée/sortie").

XCas :

> print (" `texte :` " + a) ; ou print (" `texte :` " , a) ;
> afficher (" `texte :` " + a) ; ou afficher (" `texte :` " , a) ;
> *affichent le texte "texte : " suivi de la valeur de la variable a.*

Python :

> print (" `texte` " , a)
> *affiche le texte suivi de la valeur de la variable a.*

4.2 Commandes d'entrée de valeurs

Définition :

> Les commandes d'entrée de valeurs permettent à l'algorithme de demander à l'utilisateur un nombre, un caractère ou un texte.

Syntaxe en algorithmique :
`Saisir` *a* ou `Lire` *a* ou `Entrer` *a*

Casio :

? → A
demande à l'utilisateur d'entrer la valeur à stocker dans la variable A.

? est accessible à partir de l'éditeur de programmes en faisant défiler avec la touche ▶ puis en utilisant la touche Fn qui correspond (F1 sur Graph25).

TI :

Prompt A ou Input A
demande à l'utilisateur d'entre une valeur pour la variable A.

Prompt et Input sont accessibles dans le menu PRGM I/O.
À noter, sur TI, la commande Input " texte " ,A affiche le texte entre guillemets et demande d'entrer la valeur de A.

XCas :

input (" Entrer a : " , a) ; ou saisir (" Entrer a : " , a) ;
demande à l'utilisateur d'entrer une valeur pour la variable a et attend que la valeur soit entrée.

Python3 :

a = input (" Entrer a : ")
demande à l'utilisateur d'entrer une valeur pour la variable a et attend que la valeur soit entrée. Noter que la valeur sera considérée comme une chaîne de caractères. Pour une valeur numérique, on écrira a=int(input("Entrer a : "))

5 Exercices sur les entrées et sorties

Exercice 1 :

Que fait l'algorithme suivant ?

Variables a, b, c, d : *nombres réels*
Début traitement
  saisir a ;
  saisir b ;
  stocker $a \times b$ **dans** c ;
  stocker $2 \times (a + b)$ **dans** d ;
  afficher c ;
  afficher d ;
Fin

Exercice 2 :

Que fait l'algorithme suivant ?

Variables r, d, a : *nombres réels*
Début traitement
  saisir d ;
  stocker $\frac{d}{2}$ **dans** r ;
  stocker $2 \times (a + b)$ **dans** d ;
  stocker $3,14 \times r^2$ **dans** a ;
  afficher a ;
Fin

Exercice 3 :

Écrire un algorithme qui demande d'entrer deux nombres entiers a et b et calcule le reste de la division euclidienne de a et b. On utilisera pour cela la fonction partie entière `int a` qui donne la partie entière d'un nombre a (menu MATH NUM iPart sur TI, menu OPTN NUM Int sur Casio et iPart sur XCAS).

Exercice 4 :

Écrire un algorithme qui demande d'entrer un nombre puis affiche son image par la fonction f définie par $f(x) = 3x^2 + 5x - 9$.

Exercice 5 :

1. Écrire un algorithme qui convertit des secondes en heures, minutes et secondes.
2. Écrire un algorithme qui convertit des heures en jours et heures.

Exercice 6 :

Écrire un algorithme qui demande d'entrer trois nombres A, B et C et calcule et affiche leur moyenne non pondérée.

Exercice 7 :

Écrire un algorithme qui, l'utilisateur ayant entré le taux annuel d'épargne en pourcentage et le capital initialement placé, calcule et affiche le capital disponible auquel sont ajoutés les intérêts de l'année.

6 Structures conditionnelles

6.1 Si..alors..sinon

Définitions :

Ces instructions permettent de tester si une condition est vraie ou fausse et de poursuivre le programme d'une manière différente selon que la condition est vraie ou fausse.

Syntaxe en algorithmique :

Si
condition
Alors
instructions si condition vraie
Sinon
instructions si condition fausse
FinSi

TI :

If condition
Then
instructions
Else
instructions
End

If, *Then*, *Else* et *End* sont accessibles dans le menu *PRGM* *CTL* (contrôle).

Casio :

If condition
Then
instructions
Else
instructions
ifEnd

If, *then*, *Else* et *IfEnd* sont accessibles dans le menu *SHIFT* *PRGM* puis *COM* (*F1* sur Graph25).

XCas :

si
(condition)
{ instruction ;
...
instruction ; }
sinon
{ instruction ;
...
instruction ; } ;

Python :

```
if
( condition ) :
        instruction
        ...
        instruction
else :
        instruction
        ...
        instruction
```

Attention : les tabulations (espaces) en début de ligne sont indispensables en python.

Exemple :

TI :

input "A =? ",A
If $A<7$
Then
$A+1\to A$
Else $A-1\to A$
End Disp A

Casio :

?$\to A$
If $A<7$
Then
$A+1\to A$
Else
$A-1\to A$
ifEnd "A"

XCas :

```
saisir("a = ? ",a);
si (a<7)
    {a:=a+1;}
sinon
    {a:=a-1;};
afficher("a = "+a);
```

Python :

```
a=int(input("Entrer a : "))
if (a<7):
    a=a+1
else
    a:a-1
print("a = ",a)
```

Ce programme teste si la variable a entrée a une valeur inférieure à 7 et, si c'est le cas, ajoute 1. Sinon, il enlève 1 à la valeur de la variable. Puis, quelle que soit la valeur de a, il affiche le contenu de la variable a. On remarquera les espaces laissés au début de certaines lignes pour Xcas et Maxima : ils ne sont pas indispensables mais aident à clarifier la compréhension du programme, c'est donc une bonne habitude à prendre que de les utiliser.

6.2 Opérateurs relationnels et logiques

Définition :

Pour tester une condition on utilise les *opérateurs relationnels* suivants :
- $a = b$ teste l'égalité de a et de b ;
- $a < b$ teste si a est strictement inférieur à b ;
- $a \leq b$ teste si a est inférieur ou égal à b ;
- $a > b$ teste si a est strictement supérieur à b ;
- $a \geq b$ teste si a est supérieur ou égal à b ;
- $a \neq b$ teste si a est différent de b.

On utiliser ausi pour les conditions plus complexes les opérateurs logiques "et" ("AND"), "ou"("OR") et "non" ("not").

Casio :
Les opérateurs relationnels se trouvent dans MENU PRGM ▶ REL.

TI :
Les opérateurs relationnels se trouvent dans 2nd TEST TEST et les opérateurs logiques dans LOGIC.

XCas :
- $a == b$ teste l'égalité de a et de b ;
- $a < b$ teste si a est strictement inférieur à b ;
- $a <= b$ teste si a est inférieur ou égal à b ;
- $a > b$ teste si a est strictement supérieur à b ;
- $a >= b$ teste si a est supérieur ou égal à b ;
- $a! = b$ teste si a est différent de b ;
- `condition1 && condition2` teste si les deux conditions sont vraies simultanément ;
- `condition1 || condition2` test si l'une au moins des deux conditions est vraie ;
- `!condition` teste si la négation de la condition est vraie.

Python :
- $a == b$ teste l'égalité de a et de b ;
- $a < b$ teste si a est strictement inférieur à b ;
- $a <= b$ teste si a est inférieur ou égal à b ;
- $a > b$ teste si a est strictement supérieur à b ;
- $a >= b$ teste si a est supérieur ou égal à b ;
- $a! = b$ teste si a est différent de b ;
- `condition1 and condition2` teste si les deux conditions sont vraies simultanément ;
- `condition1 or condition2` test si l'une au moins des deux conditions est vraie ;
- `!(condition)` teste si la négation de la condition est vraie.

7 Exercices sur les structures conditionnelles

Exercice 1 :

Écrire un programme qui demande l'âge de l'utilisateur et répond "vous êtes mineur" ou "vous êtes majeur" suivant le cas.

Exercice 2 :

Écrire un programme qui demande la température extérieure en degrés celsius et affiche "il gèle" si le nombre est négatif et "alerte à la canicule" si le nombre est supérieur à 30.

Exercice 3 :

1. Qu'affiche l'algorithme suivant ?

```
1000->tirelire
19->âge
Si (âge >=19 et tirelire >=1000)
alors afficher "Vous pouvez ouvrir un compte"
sinon afficher "pas de compte possible"
```

2. Écrire le code correspondant à l'algorithme précédent pour la calculatrice ou pour XCas.

Exercice 5 :

Écrire un algorithme qui, à partir d'un nombre entré par l'utilisateur, affiche ce même nombre s'il est positif et son opposé s'il est négatif (le nombre obtenu est appelé la valeur absolue du nombre entré).

Exercice 6 :

Écrire un algorithme qui, à partir de la donnée de la longueur de chacun des trois côtés d'un triangle, teste si le triangle est rectangle.

8 Boucles

Définition :

Les boucles sont utilisées pour qu'une séquence d'instructions soit répétée un nombre donné de fois ou tant qu'une condition n'est pas remplie.

8.1 Boucles "pour"

Définition :

Ces instructions sont utilisées pour contrôler les boucles en incrémentant (augmentant) une variable. La variable est augmentée d'une valeur de départ jusqu'à une valeur d'arrivée d'un pas donné (l'incrément).

Syntaxe :
Pour
`variable`
allant de
`valeur de départ`
à
`valeur d'arrivée`
faire
`instructions`
fin

Casio :

For
`valeur de départ→variable` To `valeur d'arrivée` Step `incrément instructions`
Next

Les instructions *For*, *To*, *Step*, *Next* *se trouvent dans* *SHIFT* *PRGM* *COM*.

TI :

For (`variable` , `valeur de départ` , `valeur d'arrivée` , `incrément`)
`instructions`
End

Les instructions *For*, *End* *se trouvent dans le menu* *PRGM* *CTL*.

XCas :

```
pour variable de valeur de départ jusque
        valeur finale pas valeur de l'incrément faire
instruction ;
instruction ;
...
instruction ;
fpour ;
```

Python :

```
for variable in range ( valeur de départ , valeur finale+1 ) :
    instruction
    instruction
    ...
    instruction
```

Attention : les tabulations en début de ligne dans le bloc d'instructions sont indispensables en python.

Exemple :

Variables a *: nombre entier naturel*
Début traitement
  pour a **allant de** *0* **à** *100* **par pas de** *2* **faire**
    stocker a^2 **dans** b ;
    afficher a ;
    afficher b ;
  fin
Fin

Casio :

For 0 → A To 20 Step 2
$A * A \rightarrow B$
B ⊿
Next

TI :

For (A , 0 , 20 , 2)
$A * A \rightarrow B$
Disp A , B
End

XCas :

```
pour a de 0 jusque 20 ...
      ... pas 2 faire
   b:=a^2;
   afficher(a+" : "+b);
fpour;
```

Python :

```
for a in range(0,10):
   b=(2*a)**2,
   print(a,b)
```

Cet algorithme affiche le tableau de valeurs de la fonction carré de 0 à 10 par pas de 2.

8.2 Boucles "Tant que"

Définition :

Éxécute un groupe de commandes tant qu'une condition est vraie. La condition est testée en début de boucle donc si la condition est vraie lorsqu'elle est testée en début de boucles, *toutes* les instructions contenues à l'intérieur de la boucle sont effectuées.

Syntaxe :

Tant que `condition`
`instructions`
faire
`instructions`
fin tant que

Casio :

While `condition`
`instructions`
WhileEnd

While et *WhileEnd* se trouvent dans le menu *SHIFT* *PRGM* *COM* (*F1* sur *Graph25*)

TI :

While `condition`
`instructions`
End

While et *End* se trouvent dans le menu *PRGM* *CTL*.

XCas :

tantque `condition` faire
`instruction` ;
`instruction` ;
...
`instruction` ;
ftantque

Python :

While (`condition`) :
    `instruction`
    `instruction`
    ...
    `instruction`

Tabulations nécessaires à nouveau

Exemple :

Variables a : *nombre entier naturel*
Début traitement
  stocker 10 **dans** a ;
  tant que $a > 0$ **faire**
    stocker $a - 1$ **dans** a ;
    afficher a ;
  fin
Fin

TI :

```
10 → A
While A > 0
A − 1 → A
Disp A
End
```

Casio :

```
10 → A
While A > 0
A − 1 → A
" A=" : A ⊿
WhileEnd
```

XCas :

```
a:=10;
tantque (a>0) faire
   a:=a-1;
   afficher(a);
ftantque;
```

Python :

```
a=10
while (a>0):
   a=a-1
   print(a)
```

Cet algorithme affiche le décompte de 9 à 0.

Exemple :
Algorithme de calcul du PGCD de deux nombres a et b. Il utilise la fonction partie entière $Int(a)$ qui renvoie la partie entière d'un nombre réel a.

TI :

```
Input "A=",A
Input "B=",B
1→ R
While R ≠ 0
A-B*iPart(A/B)→ R
Disp R
B→A : R→B
End
Disp "PGCD=",A
```

Casio :

```
"A=" ? →A
"B=" ? →B
1→ R
While R≠0
A-B*Int(A/B)→ R
R ⊿
B→A : R→B
WhileEnd
"PGCD=" : A ⊿
```

Variables a, b, r : *nombres entiers naturels*
Début traitement
  saisir a ;
  saisir b ;
  stocker 1 **dans** r ;
  tant que $r <> 0$ **faire**
    stocker $a - b \times Int(a/b)$ **dans** r ;
    afficher r ;
    stocker b **dans** a ;
    stocker r **dans** b ;
  fin
  afficher a
Fin

XCas :

```
saisir("a= ",a);
saisir("b= ",b);
r:=1;
tantque (r!=0) faire
r:=a-b*intDiv(a,b);
afficher("r= "+r);
a:=b;b:=r;
ftantque;
afficher("PGCD = "+a);
```

Maxima :

```
a=int(input("a= "))
b=int(input("b= "))
r=1
while(r!=0):
   r=a%b
   print("r= ",r)
   a=b
   b=r
print("PGCD =",a)
```

a%b donne le reste de la division entière de a par b.

8.3 Boucles "répéter"

Définition :

Comme les boucles "tant que", une boucle "répéter" éxécute un groupe d'instructions mais ceci jusqu'à ce que la condition soit vraie et la condition est testée en fin de boucle. Dans les deux cas, la boucle est toujours réalisée au moins une fois.

Syntaxe :
Répéter
`instructions`
jusqu'à `condition`

Casio :	**TI :**	**XCAS :**	**Python :**
Do instructions LpWhile condition *Do et LpWhile ("Loop" en anglais signifie "boucle") se trouvent dans le menu SHIFT PRGM COM (F1 sur Graph25).* *Attention sur Casio, la boucle Do LpWhile s'effectue tant que la condition est vraie et non pas jusqu'à ce que la condition soit vraie.*	repeat condition instructions End *repeat et End se trouvent dans le menu PRGM CTL.* *Attention sur TI : La condition se met au début de la boucle mais elle est testée en fin de boucle uniquement.*	*Pas de telle boucle pour XCAS, utiliser une boucle "tant que".*	*Pas de telle boucle en python, utiliser une boucle "tant que".*

Exemple :

```
Saisir A
Saisir B
1->R
Répéter
  A-B*Int(A/B) -> R
  Afficher R
  B->A
  R->B
jusqu'à R=0
Afficher "PGCD=",A
```

TI :	**Casio :**
Input "A=",A Input "B=",B 1→ R Repeat R = 0 A-B*iPart(A/B)→ R Disp R B→A :R→B End "PGCD=",A	"A=" ?→A "B=" ?→B Do A-B*Int(A/B)→ R R⊿ B→A :R→B LpWhile R ≠ 0 "PGCD=" :A⊿

Il s'agit du même algorithme d'Euclide. Observer les différences avec l'algorithme écrit à l'aide de boucles "tant que" et les différences d'écriture sur les modèles TI et Casio.

9 Exercices sur les boucles

Exercice 1 :

1. Combien de fois le message "Salut" sera-t-il affiché à partir de l'algorithme suivant ?

```
15 -> A
Répéter
  afficher "Salut"
  A+1->A
jusqu'à A<15
```

2. Combien de fois ce même message sera-t-il affiché dans le cas suivant ?

```
14->A
Tant que A<15
faire
  afficher "Salut"
finTantque
```

Exercice 2 :

1. Écrire un algorithme qui calcule la somme des nombres entiers de 0 à 50.
2. Écrire un algorithme qui calcule le produit des nombres entiers de 1 à 7
3. Écrire un algorithme qui calcule la somme des 20 premiers nombres impairs.
4. Écrire un algorithme qui calcule la somme des 20 premiers nombres paires.

Exercice 3 :

Écrire un algorithme qui calcule la variance et l'écart type d'une série de nombres entrés par l'utilisateur. L'algorithme demandera le nombre de nombres que comprend la série avant de demander d'entrer la série de nombres.

Exercice 4 :

Écrire un algorithme qui, une somme initiale ayant été demandée à l'utilisateur ainsi qu'une durée de placement en année et un taux de placement en pourcentage à intérêts composés, affiche la somme disponible au bout de la durée de placement.

Exercice 5 :

Écrire un algorithme permettant le calcul du PGCD de deux nombres entrés par l'utilisateur par la méthode des différences successives (on rappelle que les différences successives consistent à faire la différence du plus grand nombre par le plus petit et à garder la différence et le plus petit nombre à chaque étape pour recommencer jusqu'à obtention de 0).