**précédent:** Introduction **monter:** UE33/UE43 - Opt1. Algorithmique D.E.U.G. 2ème Année **suivant:** Équivalence langage algorithmique / langage C **Table des matières**

**Sous-sections**



---

# Bases d'un langage algorithmique

Le langage algorithmique est un langage générique permettant de traiter des problèmes par concaténation d'instructions élémentaires. Il est à la base de tous les langages de programmation (enfin... tous les langages de programmations *impératifs*).

## Structure de base

En matière de programmation, il n'y a pas grand chose d'obligatoire mais beaucoup de choses recommandées. En particulier, un programme a à peu près toujours la même organisation générale

```
1: Nom du programme
2: Déclaration de variables
3: Déclaration de fonctions
4: Début
5:    ...
6:    Liste des instructions
7:    ...
8: Fin
```

# D&#65533;claration de variables

Qu'est ce qu'une variable ? Une variable est un espace m&#65533;moire nomm&#65533;, de taille fix&#65533;e prenant au cours du d&#65533;roulement de l'algorithme un nombre ind&#65533;fini de valeurs diff&#65533;rentes. Ce changement de valeur se fait par l'op&#65533;ration d'**affectation** (not&#65533;e dans notre langage algorithmique). La variable diff&#65533;re de la notion de **constante** qui, comme son nom l'indique, ne prend qu'une unique valeur au cours de l'ex&#65533;cution de l'algorithme.

&#65533; quoi sert la d&#65533;claration de variable ? La partie *d&#65533;claration de variable* permet de sp&#65533;cifier quelle seront les variables utilis&#65533;es au cours de l'algorithme ainsi que le type de valeur qu'elles doivent respectivement prendre. Il est bien &#65533;vident que l'on ne peut m&#65533;moriser une cha&#65533;ne de caract&#65533;res dans une variable de type ``Entier''. Le type des variables est utile &#65533; l'algorithmicien pour lui permettre de structurer ses id&#65533;es. Il est tr&#65533;s utile au programmeur car il permet de d&#65533;tecter des erreurs potentielles. Il est indispensable au compilateur car les diff&#65533;rents types ne sont pas tous repr&#65533;sent&#65533;s de la m&#65533;me fa&#65533;on. La repr&#65533;sentation d'un m&#65533;me type peut m&#65533;me varier d'un processeur &#65533; l'autre.

On utilisera diff&#65533;rents types de variables (pour le moment):

- **Entier** (1,2,3, ...)
- **R&#65533;el** (ou flottants) (1.0, 1.35, 1E+12, 3.1415926, ...)
- **Caract&#65533;re** (a, b, c, !, ?, ...)
- **Bool&#65533;en** (VRAI ou FAUX, par exemple $(2 < 3)$ **Ou** $(3 \neq 5)$ vaut VRAI)

Par exemple, "**Entier** $A, B$"
d&#65533;finit 2 variables de type enti&#65533;res n'ayant aucune valeur (pour l'instant).

# Instruction

Une instruction est une action &#65533;l&#65533;mentaire commandant &#65533; la machine un calcul, ou une communication avec un de ses p&#65533;riph&#65533;riques (Entrant ou Sortant). Une instruction de base peut &#65533;tre :

**une affectation et/ou op&#65533;ration arithm&#65533;tique:**
l'affectation est l'action &#65533;l&#65533;mentaire principale puisque c'est par son interm&#65533;diaire que l'on peut modifier la valeur d'une variable. L'affectation a pour syntaxe variablevaleur.

```
1: { Exemple d'affectation }
2: Entier A
3: Début
4:    A← 1
5: Fin
6: {Cette séquence d'instructions donne la valeur entière "1" à la variable
   A.}
7: {Remarque : la valeur donnée est toujours du même type que la va-
   riable.}
```

**un affichage:**

l'affichage est l'action élémentaire permettant a l'utilisateur de fournir un ou plusieurs résultats issus de son algorithme. Ainsi l'affichage peut être une simple phrase mais aussi peut permettre la visualisation du contenu (typé) d'une variable. L'affichage dans le langage algorithmique se fait par l'intermédiaire de la commande ***écrire***.

```
1: { Exemple d'affectation }
2: Entier A
3: Début
4:    A← 1
5: Fin
6: {Cette séquence d'instructions donne la valeur entière "1" à la variable
   A.}
7: {Remarque : la valeur donnée est toujours du même type que la va-
   riable.}
```

Cette séquence d'instructions affiche la phrase suivante à l'écran:

```
La Valeur de A est 3.
La Valeur de A+1 est 4.
```

Dans le langage simple que nous utilisons pour écrire nos algorithmes, on indique directement ce qui doit être écrit en les séparant par des virgules. En C, le mécanisme est un peu différent. Le premier argument est une sorte de texte à trou que les autres arguments viennent remplir.

**une lecture au clavier ou dans un fichier:**

la lecture au clavier est l'action élémentaire permettant de spécifier par une intervention humaine la valeur d'une variable. Cette action symbolise donc la communication avec un périphérique d'entrée tel que le clavier. Bien évidemment, la valeur saisie par l'utilisateur de l'algorithme se d'être du même type que la variable recevant la valeur. La saisie se fait par l'intermédiaire de la commande ***Lire***.

```
1: {Exemple d'utilisation de Lire }
2: Entier A
3: Début
4:    Lire A
5:    Écrire ('La valeur de 2*A est ', 2 × A)
6: Fin
```

Si l'utilisateur saisit la valeur 10, nous aurons alors à l'écran:

```
La Valeur de 2*A est 20
```

## Savoir déchiffrer une séquence d'instructions

**Exercice 1** Que fait la liste d'instructions suivantes ?

1: $A \leftarrow 2$
2: $A \leftarrow A + 2$
3: $B \leftarrow A \times 2 + A$
4: $C \leftarrow 4$
5: $C \leftarrow B - C$
6: $C \leftarrow C + A - B$
7: $A \leftarrow B - C \times A$
8: $A \leftarrow (B - A) \times C$
9: $B \leftarrow (A + C) \times B$

Voir rponse 1.

**Exercice 2** Que fait la liste d'instructions suivantes ?

1: $X \leftarrow -5$
2: $X \leftarrow X^2$
3: $Y \leftarrow -X - 3$
4: $Z \leftarrow (-X - Y)^2$
5: $X \leftarrow -(X + Y)^2 + Z$
6: $Y \leftarrow Z^X \times Y$
7: $Y \leftarrow -(Z + Y)$
8: $X \leftarrow X + Y - Z$
9: $Y \leftarrow X + Z$
10: $X \leftarrow (Y - Z)^2$
11: $Y \leftarrow X - Y$
12: $X \leftarrow (Y + Z)/(X/10)$
13: $Y \leftarrow ((X \times Z)/Y) \times 9$

Voir rponse 2.

## Comprendre les principes de l'affectation

**Exercice 3** Comment inverser le contenu de deux variables ?

1: {Inversion de deux variables}
2: **Entier** $A, B, X$
3: **Début**
4: **Lire** $A$
5: **Lire** $B$
6: . . .
7: . . .
8: . . .
9: **Écrire** ('La valeur de A est ', $A$)
10: **Écrire** ('La valeur de B est ', $B$)
11: **Fin**

Voir rponse 3.

**Exercice 4** Comment faire sans utiliser une variable supplmentaire?

```
1:  {Inversion de deux variables}
2:  Entier A, B, X
3:  Début
4:     Lire A
5:     Lire B
6:     ...
7:     ...
8:     ...
9:     Écrire ('La valeur de A est ', A)
10:    Écrire ('La valeur de B est ', B)
11: Fin
```

Voir rponse 4.

**Exercice 5** tant donn e X, comment calculer le plus rapidement possible $X^{16}$ ?

```
1:  {Exponentiation}
2:  Entier X
3:  Début
4:     Lire X
5:     ...
6:     ...
7:     ...
8:     ...
9:     Écrire ('La valeur de X^16 est ', X)
10: Fin
```

Voir rponse 5.

### R solution de probl mes simples

**Exercice 6** crire un algorithme saisissant le prix "TTC" d'une marchandise et affichant le prix "Hors Taxe" sachant que cet article a une T.V.A. de 18,6%. Voir rponse 6.

**Exercice 7** crire un algorithme saisissant 2 variables enti res qui calcule et affiche leur moyenne. Voir rponse 7.

**Exercice 8** crire un algorithme saisissant un temps en seconde que l'on transcrira en jours, heure, minutes, secondes. Voir rponse 8.

**Exercice 9** En se basant sur l'exercice pr c dent, crire un algorithme permettant de faire la diff rence entre deux horaires saisis en heure, minutes, secondes. Voir rponse 9.

## Les conditions

La condition en algorithmique est une instruction de branchement permettant de d cider, dans un contexte donn , quelle sera la s quence d'instructions a appliquer. Elle permet ainsi  l'algorithme de prendre des d cisions concernant son ex cution. Sa syntaxe est :

```
1: ...
2: Si <condition> Alorsdefault
3:     ... Instructions A ...
4: Sinondefault
5:     ... Instructions B ...
6: ...
```

La section est facultative. La partie $<$condition$><$tex2html$_{c}omment_{m}ark > 73$
est essentielle puisque c'est elle qui d&#65533;cide de l'ex&#65533;cution des instructions conditionn&#65533;es. Elle est de type ***Bool&#65533;en***. Cela signifie que:

- si $<$condition$>$
  vaut VRAI
  , seul le bloc Instructions A
  sera ex&#65533;cut&#65533;.
- si $<$condition$>$
  vaut FAUX
  , seul le bloc Instructions B
  (s'il existe) sera ex&#65533;cut&#65533;.

## Savoir Interpr&#65533;ter une condition

On s'int&#65533;resse au bloc d'instructions suivant:

```
1: ...
2: Si (C − B) = B Alorsdefault
3:     A ← A + 1
4:     C ← C + B
5:     B ← A
6: Sinondefault
7:     B ← A
8:     A ← A − 1
9:     C ← C × B
10: ...
```

**Exercice 10** Donner les valeurs des variable A, B et C &#65533; la sortie de ce bloc d'instructions:

- pour $A \leftarrow 2$, $B \leftarrow 3$, $C \leftarrow A \times B$ Voir r&#65533;ponse 10.
- pour $A \leftarrow 1$, $B \leftarrow 5$, $C \leftarrow 3$ Voir r&#65533;ponse 10.
- pour $A \leftarrow -3$, $B \leftarrow A \times A$, $C \leftarrow B - 5$ Voir r&#65533;ponse 10.
- pour $A \leftarrow 8$, $B \leftarrow 3$, $C \leftarrow A - 2$ Voir r&#65533;ponse 10.
- $A \leftarrow 10$, $B \leftarrow 1$, $C \leftarrow -B + A^2$ Voir r&#65533;ponse 10.

## Calculer les racines d'une &#65533;quation du second degr&#65533;

**Exercice 11** Saisir 3 entiers a, b, c et d�terminer dans $\mathbb{R}$ (i.e. sans solution dans les complexes) les racines de l'�quation $aX^2 + bX + c = 0$. Voir r�ponse 11.

# Les boucles

Si le langage algorithmique se limitait aux structures pr�c�dentes, nous ne pourrions pas faire grand chose de plus qu'avec une calculatrice. On pourra nottemment remarquer que lorsque l'on a un grand nombre d'op�rations similaires � faires, le programme se d�roulant de fa�on lin�aire, il est n�cessaire d'�crire ces op�rations autant de fois que n�cessaire. On introduit donc d'autres structures de contr�le : les boucles.

## Boucle finie

La boucle permet d'appliquer une op�ration (c'est-�-dire un ensemble d'instructions) � chaque �l�ment d'une liste.

```
1: Entier i
2: Début
3:    Pour i dans {1..100} :default
4:       Écrire (i^2)
5: Fin
```

Ces boucles peuvent se pr�senter sous diff�rentes formes:

```
1: Pour i dans {1..100} :default
2:    ...
3: Pour i= 1 à 100 :default
4:    ...
5: Pour i= 1 à 100 de 5 en 5 :default
6:    ...
7: Pour i=23 à 12 de -1 en -1 :default
8:    ...
```

**Exercice 12** En vous inspirant de l'exemple pr�c�dent, �crivez un algorithme qui demande un nombre $n$, calcule et affiche la somme $\sum_{i=0}^{n} i^3$. Voir r�ponse 12.

## Boucle ind�finies

La boucle finie permet donc de r�aliser des t�ches r�p�titives � l'aide d'un ordinateur. N�anmoins, il n'est pas toujours possible de d�crire simplement l'ensemble des indices que doit parcourir la boucle. Dans le cas, par exemple, o� l'on cherche le premier entier $i > 1$ tel que la $i^4 + 4$ soit premier, on ne peut (� moins de r�fl�chir un peu) pas savoir la taille de l'ensemble d'indice � observer. Peut-�tre m�me qu'un tel nombre n'existe pas...Une chose est certaine, si ce nombre existe, une boucle permettra de le trouver.

```
1: Entier i
2: Début
3:    i← 2
4:    Tant que i^4 + 4 n'est pas premier :default
5:       i← i+1
6:    Écrire ('Ce nombre existe et vaut ',i)
7: Fin
```

# Les variables indic&#65533;es (ou tableaux)

Une variable standard permet de stocker une valeur. Une variable indic&#65533;e de 1 &#65533; $n$ permet de stocker $n$ valeurs. La d&#65533;claration d'un tableau dont les &#65533;l&#65533;ments ont un type de base, a une structure tr&#65533;s proche d'une d&#65533;claration de variables ayant un type de base. La seule diff&#65533;rence consiste &#65533; indiquer entre crochets le nombre d'&#65533;l&#65533;ments du tableau apr&#65533;s le nom de la variable.

- **Entier** $t[10]$
- **Réel** $a[100]$

L'acc&#65533;s aux valeurs du tableau se fait donc &#65533; l'aides d'indices.

# Les fonctions

Dans l'exemple de la section 1.5.2, le test **Tant que** $(i^4 + 4$ n'est pas premier$)$ avait &#65533;t&#65533; utilis&#65533;. Ce n'est pas une instruction &#65533;l&#65533;mentaire et la d&#65533;termination de la valeur de " $i^4 + 4$ n'est pas premier " n&#65533;cessite un certain nombre de calculs. Il serait parfaitement possible d'ins&#65533;rer l'ensemble d'instruction correspondant avant et et au d&#65533;but du corps de la boucle . Cela nuirait cependant &#65533; la lisibilit&#65533; de l'algorithme et ne serait pas tr&#65533;s pratique. Lorsqu'un ensemble d'instructions r&#65533;alise un certain algorithme et que cet ensemble est utilis&#65533; &#65533; diff&#65533;rents endroits, il faut utiliser une fonction.

La structure d'une fonction ressemble &#65533; s'y m&#65533;prendre &#65533; celle d'un programme si ce n'est qu'elle peut prendre un certain nombre de param&#65533;tres en entr&#65533;e et qu'elle doit renvoyer une valeur.

```
type_retour NOM DE LA FONCTION(type_1 var_1, type_2 var_2, ...)
 1:     {Déclaration des variables locales}
 2: ...
 3: Début
 4:    ...
 5:    ...
 6:       {Liste des instructions}
 7:    ...
 8:    ...
 9:    Renvoyer     {une valeur de type type_retour}
10: Fin
```

Si a est de type ***R&#65533;el*** et b de type ***Entier***, il est ensuite possible d'appeler une fonction power prenant deux arguments (le premier de type ***R&#65533;el*** et le second de type ***Entier***) et renvoyant un ***R&#65533;el*** (par exemple) de la fa&#65533;on suivante: $res = \text{TOTO}(a,b)$

**Exercice 13**  titre d'exemple, comment crire une fonction qui dtermine si un nombre n'est pas premier? Voir rponse 13.

Notons quelque-chose d'extrmement important concernant les variables (mme si cette notion est plus smantique qu'algorithmique). Une variable a une porte. Elle n'est visible que dans une certaine zone. Pour mieux comprendre cette notion, regardons l'exemple suivant:

```
1:  {Porté 1}
2:  Réel i
3:
FOO()
4:  Entier i
5:  Début
6:      i← 1
7:      Écrire (i)
8:  Fin
9:
10: Début
11:     i← 3.14156926
12:     Écrire (i)
13:     FOO()
14:     Écrire (i)
15: Fin
```

La sortie de toute mise en œuvre raisonnable de l'algorithme prcdent est:

```
3.14156926
1
3.14156926
```

La variable i qui est modifie ligne 6 dans la fonction foo() est celle qui est dfinie ligne 4 et pas celle qui est dfinie ligne 2.  la sortie de la fontion foo(), i (dfinie ligne 2) n'est donc pas modifie.

Il est donc possible de dfinir des variables en cours d'excution du programme. Lorsque l'on est  l'extrieur de la fonction foo(), il n'est pas possible d'accder  la variable i dfinie ligne 4. L'algorithme suivant illustre cette impossibilit.

```
1:  {Porté 2}
FOO()
2:  Entier j
3:  Début
4:      j← 1
5:  Fin
6:
7:  Début
8:      FOO()
9:      Écrire (j)
10: Fin
```

Si l'on essayait de programmer cet algorithme tel quel, le compilateur nous dirait qu'en ligne 9, la variable j n'a pas t dclare.

---

**précédent:** Introduction **monter:** UE33/UE43 - Opt1. Algorithmique D.E.U.G. 2ème Année **suivant:** Équivalence langage algorithmique / langage C **Table des matières**

*Arnaud Legrand*
*2003-08-18*