Introduction à l'algorithmique

par PLU Julien (page perso)

Date de publication : 02 mai 2008

Dernière mise à jour :

Cet article s'adresse principalement aux personnes désirant débuter en programmation, ils apprendront ici les bases de construction des algorithmes qu'ils voudront utiliser et/ou fabriquer et surtout voir comment cela fonctionne.

Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

1 - Introduction

1-1 - D'où ça vient à quoi ça sert ?

L'algorithmique ne date pas d'hier puisque les premiers algorithmes remontent à environ 1800 ans avant J.C avec les babyloniens, ensuite Euclide (PGCD), Eratostène (suite des nombres premiers) et beaucoup d'autres. On peut aussi s'imaginer que les algorithmes ne se traitent qu'avec des nombres et bien non il existe énormément d'algorithmes qui ne concernent pas essentiellement les nombres comme l'algorithme génétique (ADN), algorithme de sortie d'un labyrinthe, algorithme de jeux (par exemple le min max très utilisé en IA) et bien d'autres encore. Les algorithmes ne se décrivent pas avec un langage de programmation contrairement aux idées reçues et donc ne nécessitent pas un ordinateur pour les écrire. Nous allons donc apprendre à résoudre des problèmes par le biais d'algorithmes et ensuite à les appliquer en 2 étapes:

- Ecriture d'un algorithme c'est à dire une méthode permettant de trouver une solution à partir des données d'un problème.
- Ecriture d'un programme qui consiste à traduire un algorithme pour une machine dans un langage de programmation donné.

1-2 - Petite définition

Un algorithme est une description finie d'un calcul qui associe un résultat à des données. Il est composé de 3 parties :

- Son nom
- sa spécification qui décrit quels sont les paramètres en entrée et quel est le résultat en sortie. Elle décrit le problème résolu par l'algorithme (la fonction résolu par l'algorithme).
- son corps décrit la démarche de résolution d'un problème dans un langage algorithmique, il fournit divers objets et instructions primitives ainsi que des moyens de les composer, mais ne nous empêche pas de faire appel à un algorithme dans un autre.

Remarque: par le terme langage il ne faut pas entendre quelque chose de normé mais d'évolutif car la syntaxe est propre à n'importe qui mais si on fait ça on a de forte chance de ne pas se faire comprendre par les autres d'où la nécessité d'utiliser les mêmes notations par pure lisibilitée pour les autres.

```
Algorithme : estPair?
Données : a appartient à N
Résultat : true si a est pair, false sinon

début
  si (a mod 2) = 0 alors
     renvoyer true;
  sinon
     renvoyer false;
  fin si
fin
```

Explications: ici modulo sert à renvoyer le reste de la division euclidienne de a par 2. La division euclidienne de a par b s'écrit d'une manière unique a=bq+r avec q le quotient et r le reste tels que:

0 <= r < b

Une fois l'algorithme écrit, on l'utilise par application à des arguments.

Appliquer un algorithme C'est comme l'application d'une fonction en mathématique

Avant de vous décrire le fonctionnement d'un algorithme voyons tout d'abord deux autres définitions qui sont celles de paramètre formel et de paramètre effectif.

Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

- paramètre formel: il s'agit de la variable utilisée dans le corps de l'algorithme (par ex: si on avait déclaré une variable dans le corps de l'algorithme estPair? ça serait un paramètre formel).
- paramètre effectif: il s'agit de la variable (ou valeur) fournie lors de l'appel d'un algorithme (par ex: dans l'algorithme estPair? "a" en est un car c'est une valeur donnée à l'algorithme pour savoir si elle est pair ou non).

Remarque: attention, certains langages, comme Perl, utilisent le terme paramètre pour paramètre formel et argument pour paramètre effectif.

On obtient sa valeur en substituant dans le corps de l'algorithme les arguments (par exemple 21) aux paramètres de l'algorithme (ici a) et en appliquant le corps substitué de l'algorithme obtenu dans l'étape précédente ; la valeur résultat est celle donnée par l'instruction renvoyer.

Exemple : exécution de estPair?(21) :

Substituer 21 à "a" dans le corps de estPair?. Si (21 mod 2) = 0 alors renvoyer true sinon renvoyer false. Le résultat de cette exécution est false.

1-3 - Traduction dans des langages de programmation

Un algorithme peut être traduit dans plusieurs langages de programmation, par exemple:

Traduction de estPair? en Scheme :

```
(define estPair? (lambda (a)
    (if (= ( modulo a 2) 0) #t #f)))
```

Traduction de estPair? en Maple :

```
estPair? := proc(a::integer)::boolean;
description "renvoie true si a est pair, faux sinon";
if (a mod 2) = 0 then
   return true;
else
   return false;
end if;
end proc;
```

Remarque: retenez surtout la syntaxe Maple, car celle de Scheme est assez complexe à comprendre, celui-ci étant donné à titre de pure exemple de traduction. Puis Maple a un avantage certain car étant un logiciel pour les scientifiques principalement, sa syntaxe a été conçue de sorte que l'on puisse justement traduire des étapes de calculs, d'évaluations etc..., simplement dans un langage compéhensible par un utilisateur non-averti de Maple, voir même quelqu'un qui ne s'en est jamais servi.

Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

2 - Langage algorithmique

2-1 - Types

Les objets manipulés par un algorithme ont un type. Un type est défini par :

- un domaine : l'ensemble des valeurs que peuvent prendre les objets du type
- un ensemble d'opérations qu'on peut appliquer aux objets du type

Voici quelques exemples de différents types existant dans la plupart des langages de programmation.

Type symbole :

- domaine : des noms (séquence de caractères) pas d'opération
- exemple : estPair?, a, sin,...

Remarque: attention, dans les différents langages existant certains sont des mots-clés d'autres pas.

Type entiers (relatifs) :

- domaine : Z
- opérations binaires classiques : Z × Z -> Z
- comme +, *, -, mod, ..., utilisées en notation infixée, et max, min, ... en notation préfixée
- opérations unaires classiques : Z -> Z
- comme abs, ...

Type réels :

- domaine : R
- opérations binaires classiques : R × R -> R
- comme +, *, -, /, ..., utilisées en notation infixée, et max, min, ... en notation préfixée
- opérations unaires classiques : R -> R
- comme log, sin, cos, ...

Type booléens :

- domaine : Bool={ true, false }
- opérations dont les règles sont définies dans la table :

a	b	a et b	a ou b	non (a)
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Remarque: Ce type n'existe pas dans le C inférieur à la norme C99 (et d'autres).

On utilise également des opérateurs de comparaison dont la signature est :

Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

- =, >=,... : R × R -> Bool

D'autres sont définis sur les entiers et ont pour signature :

- =, >=,... : Z × Z -> Bool

Exemple: (2 > 3) a pour valeur false.

Remarque: les différences de type de données s'expliquent par le fait que des valeurs numériques (ou autre) selon leur taille ont des besoins de mémoire différents et que les opérations mathématiques ne s'effectuent pas de la même façon selon le type de variable. Et puis on ne va pas comparer un caractère avec un nombre ça n'a aucun sens.

2-2 - Environnement

On appelle environnement un ensemble d'associations nom-valeur (symbole valeur). L'environnement par défaut est formé des symboles prédéfinis du langage d'algorithme qui sont conventionnellement : les noms des opérateurs, fonctions et constantes prédéfinis (ceux des types de base). Un environnement peut être enrichi ou modifié en affectant une valeur à une variable

Une variable a un nom (un symbole), un type, et éventuellement une valeur connue ou non, pour commencer attribuez toujours une valeur à vos variables pour que vous soyez sur de ce que vous faite, sa valeur peut varier au cours de l'exécution de l'algorithme. Une variable doit être déclarée par une instruction de la forme : nom : type;

Exemples : a : Entier; test : Booléen; b : Réel;

Une variable déclarée n'a pas de valeur. La valeur d'une variable est définie/modifiée par une instruction d'affectation.

2-3 - Expression

Avec les constantes, les opérations et les fonctions de base, les algorithmes que nous avons définis et les variables, on construit des expressions. Une expression est définie par:

- une constante (une valeur du domaine d'un type de base) : 7
- le nom d'une opération ou fonction de base : -
- le nom d'une variable : x
- le nom d'un algorithme : estPair?
- l'application d'une opération de base : (6 * 4)
- l'application d'une fonction ou d'un algorithme : estPair?(21), sin(90)

Exemple (h et i étant des noms de variable):

- h
- estPair?(9,2)
- (5 + (2 * 9))
- (true et true)

sont toutes des expressions sémantiques correctes (le terme de sémantique est utilisé comme inverse de syntaxique).

2-4 - Evaluation

Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

La valeur Val(exp) d'une expression dans un environnement Env est définie par :

Au cas où exp est	Val(exp) dans Env renvoie
une constante	la constante
un symbole nom de variable	Renvoyer la valeur de la variable dans Env
l'application d'une opération: "(exp1) op (exp2)"	Évaluer v1 = Val(exp1) et v2 = Val(exp2). Renvoyer l'application de op aux opérandes v1 et v2
l'application d'une fonction: f (exp1>, ..., (expn)	Évaluer dans l'environnement.Env v1 = Val(exp1),. . ., vn = Val(expn). Renvoyer l'application de f aux arguments v1,. . .,vn
l'application d'un algorithme: algo(exp1, ..., expn)	Évaluer dans l'environnement Env v1 = Val(exp1),. . ., vn = Val(expn). algo a pour paramètres x1,. . .,xn et un corps C. Remplacer dans C chaque paramètre par la valeur calculée précédemment. La valeur renvoyée est celle du corps ainsi substitué.
Autres cas	Renvoyer Erreur. Par exemples quand le nombre de paramètres de la fonction ou de l'algorithme ne correspond pas au nombre d'arguments. Ou bien quand le type d'un argument n'est pas celui du paramètre. Ou bien si le symbole nom de variable est celui d'une variable qui n'a pas de valeur dans Env.

Exemple en supposant que dans l'environnement courant:

- Les variables x, som et h ont la valeur 7, 5 et true
- la variable i n'est pas déclarée

Alors:

- Val((2 + (6 * 4))) revoie 26
- Val(max((15-x),som)) renvoie 8
- Val((1+i)) renvoie erreur
- Val((h et false)) renvoie false
- (estPair?(7,2,x)) renvoie erreur

2-5 - Affectation et séquence

La syntaxe d'une affectation est nomVariable := expression. Les actions réalisées lors de son exécution sont :

- On évalue expression dans l'environnement courant
- Si cette évaluation revoie erreur, l'affectation n'est pas exécutée, l'exécution générale se termine (il faut éviter ce genre de situation !). Sinon, soit E la valeur Val(expression)
- Si la variable n'a pas été déclarée, l'affectation n'est pas exécutée, l'exécution générale se termine (il faut éviter cela !)
- Si la variable a été déclarée d'un type différent du type de E, l'exécution générale se termine (il faut éviter cela !)
- Sinon, l'environnement est modifié : la valeur de la variable devient E

Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Exemple a,b sont deux variables déclarées de type Entier:

- a := 4 renvoie Val(a)=4
- a := (b+2) renvoie Val(a)=Val(b)+2

On considèrera que l'ordre d'évaluation des sous-expressions d'une expression n'a pas d'importance (Val(a + b) = Val(b + a)). Exception pour les opérateurs booléens "et" et "ou".

Lors de l'évaluation de l'expression (a et b) :

- on calcule Val (a)
- Si Val(a) = false alors Val((a et b))=false on n'évalue pas b (on parle alors d'évaluation paresseuse)
- Si Val(a) = true on calcule Val(b) ;Val((a et b))=Val(b)

Lors de l'évaluation de l'expression (a ou b) :

- on calcule Val(a)
- Si Val(a) = true alors Val((a ou b))=true on n'évalue pas b (idem que pour le "et")
- Si Val(a) = false on calcule Val(b) ;Val((a ou b))=Val(b)

Conséquence : (a et b) et (b et a) n'ont pas nécessairement la même valeur.

Exemple

Val(a)	Val((a>0)et(log(a)=20))	Val((log(a)=20)et(a>0))
2	false	false
0	false	provoque une erreur

Le corps d'un algorithme est une ou plusieurs instructions. L'instruction de base est l'affectation. On peut composer ces instructions pour définir de nouvelles instructions. Il existe plusieurs types de composition, appelés structures de contrôle. La séquence d'instructions Inst1,Inst2,...,Instn s'écrit Inst1 ;Inst2 ;... ;Instn. L'exécution de cette instruction a pour effet d'exécuter Inst1, puis Inst2 ..., puis enfin Instn.

2-6 - Instructions conditionnelles

2-6-1 - Si simple

```
si Cond alors
   Inst;
fin si
```

Où Cond est une expression à valeur booléenne lors de son exécution, l'expression Cond est évaluée. Si Val(Cond)=true, l'instruction Inst est exécutée, sinon rien n'est exécuté.

2-6-2 - Si alors sinon

```
si Cond alors
   Inst1;
sinon
   Inst2;
fin si
```


Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

où Cond est une expression à valeur booléenne lors de son exécution, l'expression Cond est évaluée. Si Val(Cond)=true, l'instruction Inst1 est exécutée, sinon l'instruction Inst2 est exécutée.

2-6-3 - Si alors sinon si

```
si Cond alors
   Inst1;
sinon si Cond2 alor
   Inst2;
sinon
   Inst3;
fin si
```

Où Cond1 et Cond2 sont des expressions à valeur booléenne lors de son exécution, l'expression Cond1 est évaluée. Si Val1(Cond1)=true, l'instruction Inst1 est exécutée, sinon, l'expression Cond2 est évaluée. Si Val(Cond2)=true, l'intruction Inst2 est exécutée, sinon l'instruction Inst3 est exécutée.

2-7 - Itérations

2-7-1 - Itération pour simple

i étant une variable de type entier (ou réel) déclarée, E2 une expression à valeur entière supérieure ou égale à 1, l'instruction répétitive Pour s'écrit:

```
pour i de 1 à E2 faire
     Inst;
fin pour
```

- Soit V2 la valeur E2
- Exécuter l'affectation i := 1
- Soit Vi la valeur de i, si Vi > V2 l'itération s'arrête
- Sinon (V2 >= Vi) exécuter l'instruction Inst puis exécuter l'affectation i := i+1 puis recommencer en 1

Remarque: E2 est évaluée autant de fois qu'il y a de passage dans la boucle, et il ne faut pas la modifier car imaginons cette situation où la boucle serait sans fin:

```
N := 2;
pour i de 1 à N
     N := N + 1;
fin pour
```

Exemple:

```
Algorithme : multiplier par 5
Données : n appartient à N
Résultat : 5.n

S, i: Entier;
début
  S := 0;
  pour i de 1 à 5 faire
       S := S + n;
  fin pour
  renvoyer S;
fin
```

2-7-2 - Itération pour généralisée

Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

i étant une variable de type entier déclarée, E1, E2, E3 3 expressions à valeur entière, l'instruction répétitive pour s'écrit:

```
pour i de E1 à E2 par pas de E3 faire
     Inst;
fin pour
```

- Evaluer les expressions E1, E2, E3, soient V1, V2, V3 leurs valeurs
- Exécuter l'affectation i := V1
- V3 est supposée positive
- Soit Vi la valeur de i, si Vi > V2 l'itération s'arrête
- Sinon (V2 >= Vi) exécuter l'instruction Inst puis exécuter l'affectation i := i+V3 puis recommencer en 1

On remarquera que E3 est optionnelle puisque par défaut, si il est omit de préciser par pas de E3, elle vaut 1. Dans une boucle pour **on ne doit pas** modifier la valeur de i, E1, E2 et E3. On notera aussi qu'une fois la boucle finie la variable i n'a aucune valeur.

Exemple:

```
Algorithme : somme
Données : n appartient à N
Résultat : somme i

S, i: Entier;
début
  S := 0;
  pour i de 1 à n faire
       S := S + 1;
  fin pour
  renvoyer S;
fin
```

2-7-3 - Itération tant que

```
tant que Cond faire
     Inst;
fin tant que
```

Où Cond est une expression booléenne. L'exécution d'un tant que revient à:

- évaluer Cond
- si Cond vaut false la boucle s'arrête sinon on exécute Inst et on recommence à 1

On remarquera qu'à la différence de la boucle pour, Inst doit modifier la valeur de Cond et le nombre d'itération dépend de l'instruction itérée.

Exemple:

```
Algorithme : puissance de 2?
Données : n appartient à N
Résultat : true si m vaut 1 => (n est une puissance de 2), false sinon

m: Entier;
début
  m := n;
  tant que (m mod 2) = 0 faire
       m := m/2;
  fin tant que
```


Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

```
  renvoyer (m = 1);
fin
```

On remarquera que l'on ne divise pas n par 2 car **on a pas le droit de modifier les paramètres** , c'est pour cela qu'on lui attribue une variable propre pour le faire, c'est à dire qu'il faut les affecter à une variable et c'est cette variable qu'il faut modifier.

Conseil: utilisez un pour quand vous connaissez le nombre d'itérations maximale et un tant que quand vous ne le connaissez pas, mais vous pouvez voir que pour et tant que sont vraiment liées car la boucle pour quand on l'implémente n'est rien d'autre qu'une boucle tant que (implémenter: programmer une fonction donnée).

Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

3 - Tableaux

3-1 - Définition tableau

Un tableau est une structure de données de base qui est un ensemble d'éléments (des variables ou autres entités contenant des données), auquel on a accès à travers un numéro d'index (ou indice). Lorsque l'on déclare un tableau on donne à la variable un nom, une taille non nulle et un type qui sera celui de tout ses éléments. Ex: T:array 1..8 of Réel; dans cette exemple on a déclaré un tableau de taille 8 contenant des réels. On accède aux éléments du tableau par leur indice (T[i]).

La déclaration d'un tableau ne fournit pas de valeur pour chacun de ses éléments. Ils doivent être initialisés par une instruction d'affection.

Exemple:

```
T:array 1..8 of Réel;
T[1]:=5.4;
T[2]:=7.2;
T[3]:=0.8;
T[8]:=7.0;
```

On remarque ici que l'on a déclaré un tableau de taille 8 contenants des réels, et que l'on a seulement initialisé les indices 1, 2, 3 et 8 donc si on appelle T[4] cela nous retournera une erreur, en fait non pas vraiment mais c'est pas très logique de faire ça.

3-2 - Tableau comme paramètre d'un algorithme

Exemple:

```
Algorithme : sommeTableau
Données : T tableau d'entiers initialisé
Résultat : somme des éléments du tableau

i,x: Entier;
début
  x:=T[1];
  pour i de 2 à taille(T) faire
       x := x+T[i];
  fin pour
  renvoyer x;
fin
```

3-3 - Tableau comme résultat d'un algorithme

Exemple:

```
Algorithme : remplirTableau
Données : n appartient à N
Résultat : tableau des n premiers entiers

i: Entier;
T:array 1..n of Entier;
début
  pour i de 1 à n faire
       T[i] := i;
  fin pour
  renvoyer T;
fin
```


Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

On peut aussi avoir un algorithme qui prend en paramètre un tableau et qui renvoi un tableau.

Exemple:

```
Algorithme : sommeDe2Tableaux
Données : T et S tableaux d'entiers initialisé de même taille
Résultat : somme de deux tableaux

i: Entier;
C:array 1..taille(T) of Entier;
début
  pour i de 1 à taille(T) faire
       C[i] := T[i]+S[i];
  fin pour
  renvoyer C;
fin
```


Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

4 - Algorithmes de base

4-1 - Multiplication

```
Algorithme : multiplication
Données : a et b appartiennent à N
Résultat : produit ab

i,x: Entier;
début
  x:=0;
  pour i de 1 à b faire
       x := x+a;
  fin pour
  renvoyer x;
fin
```

4-2 - PGCD de 2 entiers

```
Algorithme : pgcd
Données : a et b appartiennent à N
Résultat : pgcd (a,b)

i,x,q,r: Entier;
début
  i:=a; x:=b; r:=a mod b;
  tant que r != 0 faire
       i:=x;
       x:=r;
       r:=i mod x;
  fin tant que
  renvoyer x;
fin
```

4-2-1 - Erreur sur cet algorithme

```
Algorithme : faux-pgcd
Données : a et b appartenant à N
Résultat : pgcd(a,b)

q,r: Entier;
début
  r:=a mod b;
  tant que t != 0 faire
       a:=b;
       b:=r;
       r:=a mod b;
  fin tant que
  renvoyer b;
fin
```

Attention, on n'a pas le droit de modifier les paramètres données à l'algorithme

4-3 - Minimum d'un tableau d'entier

```
Algorithme : minTableau
Données : T tableau d'entier initialisé
Résultat : min de T

i,x: Entier;
début
  x := T[1];
  pour i de 2 à taille(T) faire
```


Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

```
        si x > T[i] alors
           x := T[i];
        fin si
  fin pour
  renvoyer x;
fin
```

4-3-1 - Erreur

```
Algorithme : faux-minTableau
Données : T tableau d'entier initialisé
Résultat : min de T

i,x: Entier;
début
  x := 0;
  pour i de 2 à taille(T) faire
        si x > T[i] alors
           x := T[i];
        fin si
  fin pour
  renvoyer x;
fin
```

Ici la faute est d'avoir initialisé x à 0 car, dans ce cas, le test sera toujours faux (le tableau ne contient que des entiers >= 0)

4-4 - Recherche d'un élément dans un tableau

```
Algorithme : rechTableau
Données : T tableau d'entier initialisé, x appartenant à N
Résultat : renvoi true si l'élément est dans le tableau false sinon

i: Entier;
début
  pour i de 1 à taille(T) faire
        si T[i] = x alors
           renvoyer true;
        fin si
  fin pour
  renvoyer false;
fin
```

4-4-1 - Erreur

```
Algorithme : faux-rechTableau
Données : T tableau d'entier initialisé, x appartenant à N
Résultat : renvoi true si l'élément est dans le tableau false sinon

i: Entier;
début
  pour i de 1 à taille(T) faire
        si T[i] = x alors
           renvoyer true;
        sinon
           renvoyer false;
        fin si
  fin pour
fin
```

Cet algorithme nous retournera toujours false.

4-5 - Petit exercice

Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

Petit exercice pour vous, prenez une feuille de papier et développez ce que font ces algorithmes pour comprendre leur fonctionnement, vous verrez qu'au fur et à mesure que vous allez en faire ça rentrera mieux.

Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

5 - Récursivité

5-1 - Introduction à la récurrence

La récurrence est naturellement présente en mathématiques comme en informatique. Dans les définitions d'objets, comme dans celle des fonctions. La petite différence entre les deux étant que la récurrence est une relation entre plusieurs termes successifs d'une suite, qui permet de calculer celui d'indice le plus élevé en fonction des autres et que la récursivité est une démarche qui consiste à faire référence à ce que fait l'objet dans la démarche, ainsi c'est le fait de décrire un processus dépendant de données d'un même processus sur d'autres données plus facile (condition d'arrêt).

On définit "objet" par récurrence en spécifiant:

- Les cas d'arrêt: des constantes donnant la valeur de "objet" sans faire appel à la récurrence.
- Les équations de récurrence: des définitions de "objet" qui font appel à d'autres valeur de objet

Une récurrence sera dite correcte si dans l'utilisation des équations de récurrence, la suite des appels récursifs est finie. C'est à dire si le processus d'appel de la définition, qui appelle la définition avec une deuxième valeur, qui appelle la définition avec une troisième valeur, qui...., se termine toujours.

Exemple:

On définit ExprB par récurrence, en disant, Un ExprB est soit:

- Une constante ou un symbole comme condition d'arrêt
- Une équation de récurrence (ExprB operation ExprB), où operation est un symbole d'opération binaire. Ou alors fonction(ExprB, ExprB) où fonction est un symbole de fonction à deux paramètres définie ailleurs (dans un autre algorithme par exemple).

Exemple: (par application de la définition on reconnait des objets qui sont des ExprB)

x	(5+x)	(5+(5+x))
((5+x)+5)	f(5,(5+x))	x(f,(5+x))
f(f(f(x,y),y),y)	5(x+y) est faux	(a+b+c) est faux

5-2 - Divers exemples

5-2-1 - La fonction factorielle

On définit la fonction factorielle, en disant que factorielle s'applique à un entier n appartenant à N:

- condition d'arrêt: factorielle(0) renvoie 1
- équation de récurrence: pour n > 0 factorielle(n) renvoie le résultat de n * factorielle(n-1)

La manière de spécifier ceci en mathématiques est, en utilisant la notation postfixée n! pour factorielle(n) (postfixée veut dire que l'on ajoute le symbole après la variable ou autre chose, ex: 12 min):

0! = 1	pour tout n > 0, n! = n.(n-1)!

```
Algorithme : factorielle
Données : n appartenant à N
```


Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

```
Résultat : n!

début
  si n = 0 alors
     renvoyer 1;
  sinon
     renvoyer n * factorielle(n-1);
  fin si
fin
```

Pour toute valeur de l'entier n appartenant à N le calcul de factorielle(n) termine. En effet la suite des valeurs de l'argument est la suite n, n-1, n-2,... décroissante, admettant pour minimum 0. Et on connait la valeur de factorielle(0) pour ce cas de base.

5-2-2 - La fonction pair

On définit la fonction pair, en disant, que pair s'applique à un entier n appartenant à Z:

- condition d'arrêt: pair(0) renvoie 1
- équation de récurrence: pour n > 0 pair(n) renvoie le résultat de n * pair(n-2)

```
Algorithme : pair
Données : n appartenant à Z
Résultat : vérifie si un nombre est pair

début
  si n = 0 alors
     renvoyer true;
  sinon
     renvoyer pair(n-2);
  fin si
fin
```

La définition à l'air correcte non ? et pourtant il y a un cas ou l'algorithme ne se termine jamais voyez-vous lequel ? Cela ne marche pas lorsque les chiffres sont impairs, pour que ça marche il faudrait rajouter un test pour n=1 comme ceci:

```
Algorithme : pair
Données : n appartenant à Z
Résultat : vérifie si un nombre est pair

début
  si n = 0 alors
     renvoyer true;
  sinon si n = 1 alors
     renvoyer false;
  sinon
     renvoyer pair(n-2);
  fin si
fin
```

La condition d'arrêt dans un algorithme de récurrence est ce qu'il y a de plus important, c'est la première chose à laquelle vous devez penser.

5-2-3 - Suite de Fibonacci

On définit la fonction fibo, en disant, que fibo s'applique à un entier n appartenant N:

- condition d'arrêt: fibo(0) ou fibo(1) renvoie 1

Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

- équation de récurrence: pour n > 1 fibo(n) renvoie le résultat de fibo(n-1) + fibo(n-2)

```
Algorithme : fibo
Données : n appartenant à N
Résultat : calcul la suite de fibonacci

début
  si n = 0 ou n = 1 alors
     renvoyer 1;
  sinon
     renvoyer fibo(n-1) + fibo(n-2);
  fin si
fin
```

La suite de fibonacci est représentée comme suit: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,... on peut voir que chaque terme de cette suite est la somme des deux termes précédents et se note: F(n+2) = F(n+1) + F(n).

Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

6 - Conclusion générale

6-1 - Epilogue

Apprendre les bases de l'algorithmique n'est pas une mince affaire mais cela en vaux la peine si vous voulez un jour programmez vos propres applications. Le plus important est de ne pas perdre le fil et la seule façon de vraiment assimiler tout ceci est d'en faire le plus possible pour que cela devienne un automatisme.

6-2 - Pour aller plus loin

Maintenant si vous vous sentez à l'aise avec l'algorithmique, vous passez à la pratique sur machine avec le cours sur **Maple** ou un autre langage.

6-3 - Remerciements

Remerciements à **PRomu@ld** , **bbil** , **Alp** et **pseudocode** pour leur aide à la mise en place de cet article.

Copyright © 2008 - www.developpez.com. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.