Conception de l'algorithme pour le calcul de l'épargne

On cherche à calculer le nombre d'années nécessaires pour qu'une somme de 1000 € placée à un taux annuel de 8,25 % ait atteint le double de sa valeur. Sans chercher de solution directe, on peut répéter des multiplications par 1.0825 un certain nombre de fois... et prévoir que le calcul s'arrête dès que le montant est supérieur à 2000.

Un algorithme ne s'écrit pas directement ligne par ligne "de haut en bas". Il faut réfléchir au problème posé et décider point par point de la solution que l'on va mettre en œuvre. Une fois que la solution est claire, on peut rédiger proprement l'algorithme.

Voici la résolution proposée :

1. Nous allons avoir besoin d'une variable qui stockera le montant du compte après un certain nombre d'années, appelons la `montant`, elle aura le type `réel`. Il nous faut également une variable comptant le nombre d'années écoulées, puisque c'est ce que nous cherchons. Elle sera de type entier, appelons la `nb_années`.

   Nous savons donc que notre environnement comprendra les variables suivantes :

   `montant` type réel, montant du compte à un moment donné

   `nb_années` type entier, nombre d'années écoulées

   Peut-être y en aura-t-il d'autres, nous nous en rendrons compte plus tard le cas échéant.

2. Le principe de l'algorithme est de simuler le passage des années sur notre compte bancaire. Que se passe-t-il à chaque fois qu'une année s'est écoulée ?
   - le compte reçoit les intérêts : `montant` ← `montant` × `1.0825`
   - évidemment, le nombre d'années écoulées augmente de 1 : `nb_années` ← `nb_années + 1`

   Nous venons de définir les traitements (modifications de l'environnement) qui vont être effectués de manière répétée.

3. Quel est l'état initial de l'environnement (c'est à dire, quelle est la valeur initiale de chacune des variables, avant que l'on applique les traitements répétitifs ?)
   - l'énoncé indique que le montant du compte est initialement égal à 1000 : `montant` ← `1000`
   - avant de commencer, 0 années se sont écoulées : `nb_années` ← `0`

4. Les traitements répétitifs ne doivent pas être appliqués indéfiniment, quand termine-t-on ?

   L'objectif est de savoir au bout de combien d'années le compte aura atteint la somme de 2000 €. On termine donc dès que `montant` est supérieur ou égal à 2000. Le résultat recherché, le nombre d'années écoulées, est alors dans la variable `nb_années`, dont on pourra afficher la valeur.

À ce stade, nous avons tous les éléments permettant de rédiger l'algorithme : l'environnement, sa valeur initiale, les traitements répétitifs à effectuer, la condition indiquant quand la répétition doit se terminer, l'emplacement du résultat.

Il nous reste à préciser comment répéter les traitements. Nous allons utiliser pour cela une construction `tant que`, rappelons que sa forme est :

```
tant que expression booléenne faire
      instructions
fin tant que
```

Les instructions sont les traitements que nous avons énoncés au point 2, reste à préciser la forme de l'expression booléenne. Les instructions sont exécutées *tant que* l'expression est vraie. Nous avons dit que nous devons terminer dès que `montant` ≥ `2000`, c'est à dire que nous devons continuer tant que `montant < 2000`.

Il nous reste à écrire l'algorithme au propre, sans oublier sa spécification et en plaçant éventuellement des commentaires facilitant sa lecture. Il est également très important d'utiliser correctement l'indentation pour bien faire apparaître sa structure.

```
Algorithme Épargne
(* Calcule le nombre d'années nécessaire pour doubler son épargne *)

var montant:réel         (* le montant de l'épargne *)
    nb_années:entier     (* nb d'années écoulées    *)
début
    montant ← 1000
    nb_années ← 0
    tant que montant < 2000 faire
        montant ← montant × 1.0825
        nb_années ← nb_années + 1
    fin tant que
    écrire("Montant final: ", montant, " dans ", nb_années, " an(s)")
fin
```

Une fois l'algorithme écrit, on a intérêt à faire quelques vérifications, en particulier sur la terminaison des traitements répétitifs. Peut-on se convaincre que cette répétition ne risque pas de durer indéfiniment ?

Ici on voit que la répétition se termine quand `montant` est supérieur ou égal à 2000. Sa valeur initiale est 1000, et à chaque itération elle est multipliée par 1.0825 qui est supérieur à 1. Elle augmente donc strictement à chaque fois et atteindra ou dépassera forcément la valeur 2000 après un nombre fini d'itérations.

Exécution Essayons maintenant d'exécuter l'algorithme. Nous allons pour cela faire une trace des variables. Nous nous intéressons en particulier à leur valeur avant chaque itération, au moment où la condition de continuation est évaluée.

juste avant itération n°	`montant`	`nb_années`
1	1000.00	0
2	1082.50	1
3	1171.81	2
4	1268.48	3
5	1373.13	4
6	1486.41	5
7	1609.04	6
8	1741.79	7
9	1885.49	8
10	2041.04	9

Les 2 instructions contenues dans la construction `tant que` sont exécutées 9 fois. Leur dernière exécution commence avec `montant`=1885.49 et se termine avec `montant`=2041.04. Au dernier test de la condition de continuation, `montant` vaut 2041.04 et `nb_années` vaut 9. La dixième itération n'a pas lieu, on passe à la suite et l'algorithme affiche le message :

```
    Montant final: 2041.04 dans 9 an(s)
```

Résumé de la démarche Pour écrire un algorithme il faut commencer par

– déterminer quelles vont être les données à manipuler,
– déterminer les traitements que ces données vont subir.

Les deux aspects sont complètement liés. Ici nous avons d'abord défini l'environnement (point 1), mais pour cela nous avions déjà en tête "en gros" les traitements que nous allions devoir effectuer. Parfois, en déterminant les traitements à effectuer nous nous rendrons compte que nous devons ajouter des variables à l'environnement, ou y apporter d'autres modifications.

Quand des traitements répétitifs sont en jeu (ce qui sera presque toujours le cas), il faut ensuite régler la question de la valeur initiale des variables, et de l'arrêt de la répétition.