

Quick - 1h - Correction - V3, modif en 2012

Exercice 1 - sujet 1

Écrire un algorithme qui calcule et imprime la somme des entiers de 1 à 250.

CORRECTION : On ne demande que l'impression, donc une action, qui n'a pas de paramètres. On sait que l'on fait 250 boucles, d'où l'utilisation d'une boucle pour.

```

Action Imprime()
  L: i : Entier
  Pour i de 0 à 250 Faire
    | Imprimer (i)
  Fpour
Faction

```

■

Exercice 2 - sujet 1

Soit la suite définie par $u_0 = 62$ et $u_{n+1} = 12 + u_n$ si n pair, $u_{n+1} = 1515 + u_n$ si n est impair.

1. Écrire une **fonction** qui étant donné $i \in \mathbb{N}$, calcule u_i .
2. Écrire une **action** qui imprime le premier i tel que u_i est strictement supérieur à 1 million.

CORRECTION :

1. La fonction a pour signature Entier \rightarrow Entier. Pour calculer u_i , on effectue $i - 1$ boucles, donc on connaît le nombre à l'avance, d'où l'utilisation d'une boucle pour (avec pour indice k). Attention à la parité !

```

Fonction suiteU(i) :Entier
  D: i :entier
  L: k,u : Entiers
  u  $\leftarrow$  62
  Pour k de 1 à i Faire
    | Si k mod 2 = 1 alors
      | u  $\leftarrow$  12 + u
    | Sinon
      | u  $\leftarrow$  1515 + u
    | Fsi
  Fpour
  Retourner u
Ffonction

```

2. On ne connaît pas ce i , donc on va faire une boucle tant que, le test d'arrêt est $u_i > 1000000$, donc le test de la boucle est $u_i \leq 1000000$. L'utilisation de la fonction précédente n'est pas très judicieuse (pourquoi?).

```

Action suiteUgrand(i)
  D: i : entier
  L: u : Entiers
  u ← 62
  k ← 0
  Tq u ≤ 1000000 faire
    Si k mod 2 = 1 alors
      | u ← 12 + u
    Sinon
      | u ← 1515 + u
    Fsi
    k ← k+1
  Ftq
  Imprimer k
FAction

```

■

Exercice 3 - sujet 1

Écrire un algorithme qui demande un entier à l'utilisateur et l'affiche à rebours. Par exemple, l'utilisateur saisit 123456 et le programme affiche 654321. Rappel : $153 \% 10 = 3$ et $153 / 10 = 15$.

CORRECTION : L'algorithme est une action sans argument. Soit n le nombre demandé à l'utilisateur. Pour afficher à rebours, on commence par prendre le chiffre des unités ($n \bmod 10$), puis celui des dizaines (soit $n' = n \text{ div } 10$, alors c'est $n' \bmod 10$), ... On s'arrête si $n' = 0$.

```

Action ImprimeRebours()
  L: n : Entier
  Demander(n)
  Tq n > 0 faire
    | Imprimer (n mod 10)
    | n ← n div 10
  Ftq
FAction

```

Le nombre d'opérations effectuées ici est égal au nombre de chiffres de l'écriture du nombre n , c'est $O(\log(n))$ (résultat théorique qui donne le nombre de chiffres d'un entier dans une base donnée, ici, 10). ■

Exercice 4 - sujet 1

Écrire une **fonction** qui prend un tableau d'entiers en paramètres et répond true si ce tableau est trié croissant, false sinon.

CORRECTION : La fonction prend en paramètre un tableau, et renvoie un booléen. On retourne Faux dès qu'on a trouvé une paire d'éléments dans le mauvais ordre. Attention aux indices!

Fonction *estTrie(t) : Booleen*
L: i : Entier
D: t : Vecteur[N] d'Entiers
Pour *i de 0 à N-2* **Faire**
 | Si *t[i] > t[i+1]* **alors**
 | | Retourner Faux
 | **Fsi**
Fpour
 Retourner Vrai
Ffonction

■

Exercice 5 - sujet 1

Écrire un algorithme qui calcule le schtroumpf de deux tableaux (qui ne sont pas de même taille) passés en paramètre . Pour calculer le schtroumpf, il faut multiplier chaque élément du tableau 1 par chaque élément du tableau 2, et additionner le tout. Par exemple si l'on a :

- Tableau 1 : 4 8 7 12
- Tableau 2 : 3 6

Le Schtroumpf sera :

$$3 * 4 + 3 * 8 + 3 * 7 + 3 * 12 + 6 * 4 + 6 * 8 + 6 * 7 + 6 * 12 = 279$$

Deux paramètres pourront être ajoutés : les tailles des deux tableaux d'entrée.

CORRECTION : On va écrire une fonction, et retourner l'entier calculé. La fonction prendra donc les deux tableaux tab1 et tab2, ainsi que leurs tailles. Ensuite, on utilise deux boucles pour imbriquées.

Fonction *calculeSchtroumpf(tab1, tab2, len1, len2) : Entier*
D: tab1 : Vecteur[len1] d'Entiers, tab2 : Vecteur[len2] d'Entiers
L: i, j, mult, scht : Entiers
scht ← 0
Pour *i de 0 à len2 - 1* **Faire**
 | mult ← tab1[i]
 | **Pour** *j de 0 à len1 - 1* **Faire**
 | | scht ← scht + mult * tab2[j]
 | **Fpour**
Fpour
 Retourner scht
Ffonction

■

Exercice 1 - Sujet 2

Écrire un algorithme qui calcule et imprime la somme des entiers pairs entre 9 et 1515.

CORRECTION : On ne demande que l'impression, donc une action, qui n'a pas de paramètres.

On sait que l'on fait $1515-9$ boucles, d'où l'utilisation d'une boucle pour. Pour le reste, on utilise un accumulateur qui compte la somme, et que l'on imprime à la fin.

```

Action Imprime()
  L: i : Entier
  cpt ← 0;
  Pour i de 9 à 1515 Faire
    Si i mod 2 = 0 alors
      | cpt ← cpt + 1;
    Fsi
  Fpour
  Imprimer(cpt);
FAction

```

■

Exercice 2 - Sujet 2

Soit la suite définie par $u_0 = 62$ et $u_{n+1} = 12 + n * u_n$.

1. Écrire une **fonction** qui étant donné $i \in \mathbb{N}$, calcule u_i .
2. Écrire un **action** qui imprime le premier i tel que u_i est strictement supérieur à 12 milliards.

CORRECTION :

1. La fonction a pour signature Entier \rightarrow Entier. Pour calculer u_i , on effectue $i - 1$ boucles, donc on connaît le nombre à l'avance, d'où l'utilisation d'une boucle pour (avec pour indice k).

```

Fonction suiteU(i) :Entier
  D: i :entier
  L: k,u : Entiers
  u ← 62
  Pour k de 1 à i Faire
    | u ← 12 + k * u
  Fpour
  Retourner u
FFunction

```

2. On ne connaît pas ce i , donc on va faire une boucle tant que, le test d'arrêt est $u_i > 12 \cdot 10^9$, donc le test de la boucle est $u_i \leq 12 \cdot 10^9$. L'utilisation de la fonction précédente n'est pas très judicieuse (pourquoi?).

```

Action suiteUgrand(i)
  D: i :entier
  L: k,u : Entiers
  k ← 0
  u ← 62
  Tq u ≤ 12.109 faire
    | u ← 12 + k * u
    | k ← k+1
  Ftq
  Imprimer k
FAction

```

■

Exercice 3 - Sujet 2

Écrire un algorithme qui demande un entier à l'utilisateur et qui lui donne le nombre de chiffres que contient cet entier (dans la représentation décimale). Rappel : $153\%10 = 3$ et $153/10 = 15$.

CORRECTION : L'algorithme est une action sans argument : on va imprimer le nombre de chiffres. Soit n le nombre demandé à l'utilisateur. Pour calculer le nombre de chiffres, on va diviser n par 10, puis encore par 10... , jusqu'à obtenir 0. Une variable locale cpt comptera le nombre de divisions.

```

Action CompteChiffres()
  L: n,cpt : Entiers
  Demander(n)
  cpt ← 0
  Tq  $n > 0$  faire
    | n ← n div 10
    | cpt ← cpt+1
  Ftq
  Imprimer (cpt)
FAction

```

Le nombre d'opérations effectuées ici est égal à deux fois nombre de chiffres de l'écriture du nombre n , ie $O(\log(n))$.

■

Exercice 4 - Sujet 2

Écrire une **fonction** qui prend un tableau d'entiers en paramètres ainsi qu'un entier m et qui répond true si aucun élément du tableau n'est supérieur à m , false sinon.

CORRECTION : La fonction prend en paramètre un tableau et un élément m , et renvoie un booléen. On retourne Faux dès qu'on a trouvé un élément du tableau strictement supérieur à m .

```

Fonction estTrie(t,m) :Booleen
  L: i : Entier
  D: t : Vecteur[N] d'Entiers
  Pour  $i$  de 0 à  $N-1$  Faire
    | Si  $t[i] > m$  alors
      | Retourner Faux
    Fsi
  Fpour
  Retourner Vrai
FFonction

```

■

Exercice 5 - Sujet 2

Écrire un algorithme qui calcule le produit scalaire de deux vecteurs de même taille passés en paramètre. Rappel :

$$u \cdot v = \sum_{i,j} v_i * u_j$$

La taille commune des deux vecteurs pourra être passée en paramètre.

CORRECTION :

Cette fonction n'est pas le produit scalaire !

On va écrire une fonction, et retourner l'entier calculé. La fonction prendra donc les deux tableaux tab1 et tab2, ainsi que la taille commune. Ensuite, on utilise deux boucles pour imbriquées.

Fonction *calculeFonctionExo(tab1,tab2,N) :Entier*

D: tab1,tab2 : Vecteurs[N] d'Entiers

L: i,j,mult, ps : Entiers

ps ← 0

Pour *i de 0 à N-1 Faire*

Pour *j de 0 à N-1 Faire*

 ps ← ps + tab1[i] * tab2[j]

Fpour

Fpour

Retourner ps

Fonction

Le produit scalaire c'est plutôt :

Fonction *calculeFonctionExo(tab1,tab2,N) :Entier*

D: tab1,tab2 : Vecteurs[N] d'Entiers

L: i, ps : Entiers

ps ← 0

Pour *i de 0 à N-1 Faire*

 ps ← ps + tab1[i] * tab2[i]

Fpour

Retourner ps

Fonction

Évidemment les deux versions ont été considérées correctes !

■