TD #7 ET #8 – APL1

8 octobre 2018
Erwan Kerrien

1 Pseudo-langage pour une fonction

Lorsque vous vous rendez compte que vous passez pas mal de temps à réécrire le même code ou presque, il peut être utile de penser "fonction !". Une fonction reprend en fait un bout d'algorithme pour le remplacer par une seule instruction. Le souci est qu'un ensemble de lignes prises dans un algorithme plus grand ne fonctionne pas en général : les lignes utilisent des variables qui ont été déclarées et initialisées avant et modifient la valeur de variables qui ont vocation à être utilisées après dans la suite de l'algorithme.

1.1 Exemple et travaux préparatoires

Un exemple est l'ensemble des lignes que vous écrivez à chaque fois qu'on vous demande de travailler sur un tableau. Il faut l'initialiser et pour ce faire, vous pouvez :

1. afficher un message demandant d'entrer le nombre d'éléments du tableau
2. puis saisir ce nombre et le stocker dans une variable n
3. puis, pour chaque élément du tableau
   (a) afficher un message demandant de saisir l'élément
   (b) saisir l'élément

Pour faire cela, on écrit les lignes suivantes

```
Afficher ("Entrez le nombre d'éléments du tableau : ")
Saisir (n)
Pour i allant de 1 à n faire
    Afficher ("Entrez l'élément numéro ",i," : ")
    Saisir (A[i])
FinPour
```

Dans ces lignes, 3 variables sont manipulées :

1. n est la taille du tableau, c'est un entier
2. A est le tableau, c'est un tableau d'entiers dans ce cas
3. i est une variable de boucle, c'est un entier

Le statut de ces variables n'est pas le même : A va sans doute être réutilisé par la suite de l'algorithme, tandis que n et i ne sont utiles qu'ici a priori. On dit que A est une variable de **sortie**, tandis que n et i sont des variables **locales**. On peut dès lors compléter l'algorithme ci-dessus en spécifiant le nom, le type et le statut des variables :

```
Sortie           : A[] : entier // Tableau renvoyé
Variables locales: n : entier // taille du tableau
                   i : entier // variable de boucle
Début
    Afficher ("Entrez le nombre d'éléments du tableau : ")
    Saisir (n)
    // initialise A comme un tableau contenant n fois 0
    A ← [0, 0, ..., 0]
    Pour i allant de 1 à n faire
        Afficher ("Entrez l'élément numéro ",i," : ")
        Saisir (A[i])
    FinPour
Fin
```

Vous noterez que je note les commentaires en les faisant précéder de `//` : le reste de la ligne est alors un commentaire.

1.2 Écriture de la fonction

Nous sommes alors prêts à encapsuler ce code dans une fonction. Une fonction doit pouvoir être **appelée**, c'est-à-dire utilisée comme instruction dans un algorithme. Il lui faut donc :

— un nom

— un ensemble de variables d'entrée, dont il faut connaître le type. On indiquera aussi un nom pour chaque variable afin qu'elle puisse être utilisée dans la fonction.

— un ensemble de variables de sortie, qui serviront à stocker les valeurs calculées par la fonction et que l'on souhaite pouvoir réutiliser ailleurs. Là encore, il faut un nom et un type pour chaque variable.

Ici, on va nommer la fonction : `LireTableau`. On peut remarquer qu'elle n'a pas de paramètre en entrée. On la notera donc ainsi :

Fonction *LireTableau*
  Entrée :
  Sortie : $A[]$: entier `// Tableau renvoyé`
  Variables locales: n : entier `// Taille du tableau`
    i : entier `// Variable de boucle`
  Début
    `Afficher` ("Entrez le nombre d'éléments du tableau : ")
    `Saisir` (n)
    `// initialise` A `comme un tableau contenant` n `fois 0`
    $A \leftarrow [0, 0, \ldots, 0]$
    Pour i **allant de** *1* **à** n **faire**
      `Afficher` ("Entrez l'élément numéro ",i," : ")
      `Saisir` ($A[i]$)
    FinPour
    Renvoyer A
  Fin

1.3 Instruction Renvoyer

L'instruction **Renvoyer** A indique que la variable A est prête et que son contenu peut être utilisé à l'extérieur de la fonction, par exemple en l'affectant à une autre variable dans l'algorithme qui appelle la fonction. Par exemple, dès lors que la fonction `LireTableau` a été définie, on peut l'appeler pour saisir les éléments d'un tableau, puis les afficher, grâce à l'algorithme suivant :

Variables : $T[]$: entier `// Tableau manipulé`
  n : entier `// Taille du tableau`
  i : entier `// Variable de boucle`
Début
  $T \leftarrow$ `LireTableau` ()
  $n \leftarrow$ `Longueur` (T)
  Pour i **allant de** *1* **à** n **faire**
    `Afficher` ("L'élément ",i," est ",$T[i]$)
  FinPour
Fin

Cette instruction **Renvoyer** doit systématiquement se trouver en fin de fonction. Mais on peut aussi la trouver à d'autres endroits (plusieurs fois) avant la dernière ligne. Dans ce cas, la fonction s'arrête prématurément et on en sort pour revenir au code de l'algorithme appelant. Par exemple, je n'ai pas testé si la valeur entrée pour n est valide (strictement positive). Ce n'est pas grave car la boucle qui suit ne sera alors pas exécutée. Mais si on voulait émettre un message d'avertissement,

on pourrait procéder de la sorte :

```
Fonction LireTableau
    Entrée           :
    Sortie           : A[] : entier // Tableau renvoyé
    Variables locales: n : entier // Taille du tableau
                       i : entier // Variable de boucle
    Début
        Afficher ("Entrez le nombre d'éléments du tableau")
        Saisir (n)
        Si n ≤ 0 alors
            Afficher ("Attention la taille entrée est incorrecte. Je renvoie un tableau vide.")
            Renvoyer []
        FinSi
        // initialise A comme un tableau contenant n fois 0
        A ← [0, 0, ..., 0]
        Pour i allant de 1 à n faire
            Afficher ("Entrez l'élément numéro ",i," : ")
            Saisir (A[i])
        FinPour
        Renvoyer A
    Fin
```

1.4 Remarques importantes

Il faut enfin noter les trois choses suivantes :

1. même si la variable A est utilisée dans la fonction `LireTableau`, rien ne nous oblige à employer le même nom de variable à l'extérieur. Ici, j'ai décidé d'utiliser T dans mon algorithme principal. Le nom A n'est utile quand dans le code de la fonction car il faut bien faire référence au tableau qu'on veut remplir. Mais ce qui est renvoyé est bien **le contenu de la variable** A et **c'est ce contenu qui est récupéré** pour être stocké dans T.

   De même, même si elles ont le même nom, les variables n et i de l'algorithe principal n'ont rien à voir avec les variables n et i de la fonction : le fait que ces dernières soient locales signifient qu'elles ne sont valides qu'à l'intérieur de la fonction. Similairement, le fait qu'on déclare toutes les variables locales, mais aussi d'entrée et de sortie pour la fonction lui donne un environnement d'objets qui lui sont strictement propres et qu'elle peut manipuler sans avoir nul besoin de savoir ce qui se passe dans un algorithme extérieur. De cette manière, d'une part le programmeur ou la programmeuse qui appelle cette fonction n'a pas besoin de savoir comment elle est codée. Il ou elle a juste besoin de savoir ce que cette fonction fait. D'autre part, on peut appeler cette fonction dans des algorithmes différents, sans imposer un nommage de variables à ces algorithmes.

2. on est obligés ici de faire appel à une fonction `Longueur` qui n'a pas été définie. Beaucoup de langages définissent une telle fonction mais ce n'est pas toujours le cas. Si on suppose que cette fonction n'existe pas, comme c'est le cas en C, une manière de faire est de faire renvoyer également la longueur du tableau par la fonction `LireTableau`. Cette longueur est stockée dans la variable locale n qui devient dès lors une variable de sortie. Le code de la fonction

`LireTableau` devient alors :

Fonction *LireTableau*
| **Entrée** :
| **Sortie** : $A[]$: entier `// Tableau renvoyé`
| n : entier `// Longueur du tableau`
| **Variables locales**: i : entier `// Variable de boucle`
| **Début**
| | `Afficher` ("Entrez le nombre d'éléments du tableau")
| | `Saisir` (n)
| | **Si** $n \leq 0$ **alors**
| | | `Afficher` ("Attention la taille entrée est incorrecte. Je renvoie un tableau vide.")
| | | **Renvoyer** $[], 0$
| | **FinSi**
| | `// initialise` A `comme un tableau contenant` n `fois 0`
| | $A \leftarrow [0, 0, \ldots, 0]$
| | **Pour** i **allant de** *1* **à** n **faire**
| | | `Afficher` ("Entrez l'élément numéro ",i," : ")
| | | `Saisir` ($A[i]$)
| | **FinPour**
| | **Renvoyer** A, n
| **Fin**

Et l'algorithme appelant la fonction `LireTableau` deviendra alors :

Variables : $T[]$: entier `// Tableau manipulé`
n : entier `// Taille du tableau`
i : entier `// Variable de boucle`
Début
| $T, n \leftarrow$ `LireTableau` ()
| **Pour** i **allant de** *1* **à** n **faire**
| | `Afficher` ("L'élément ",i," est ",$T[i]$)
| **FinPour**
Fin

Où on peut remarquer que deux variables sont affectées en un seul appel à `LireTableau`.

3. Enfin, il existe des fonctions particulières qui ne renvoient pas de résultat : on les appelle des **procédures**. C'est par exemple le cas d'une fonction qui ne ferait qu'afficher les valeurs d'un tableau. Il n'y a alors pas forcément d'instruction **Renvoyer**, sauf si on veut arrêter et sortir prématurément de la procédure. Cette procédure serait par exemple définie comme :

Procédure *AfficherTableau*
| **Entrée** : $A[]$: entier `// Tableau à afficher`
| n : entier `// Longueur du tableau`
| **Sortie** :
| **Variables locales**: i : entier `// Variable de boucle`
| **Début**
| | **Si** $n \leq 0$ **alors**
| | | `Afficher` ("Tableau vide")
| | **FinSi**
| | **Pour** i **allant de** *1* **à** n **faire**
| | | `Afficher` ("L'élément ",i," est ",$A[i]$)
| | **FinPour**
| **Fin**

Avec cette nouvelle procédure, l'algorithme principal devient très aisément lisible :

Variables : $T[]$: entier `// Tableau manipulé`
n : entier `// Taille du tableau`
Début
  $T, n \leftarrow$ `LireTableau` ()
  `AfficherTableau` (T,n)
Fin

2 Exercices

1. Reprenez l'exercice 2.2 du TD4 et réécrivez-le de telle manière que $f(t)$ soit calculée par un appel à une fonction appelée `Cubique`. Je vous rappelle le texte de cet exercice :
   Soit la fonction $f(t) = 2t^3 - t^2 - 37t + 36$. Écrire un algorithme qui affiche la valeur minimale et la valeur maximale prise par cette fonction sur l'intervalle $[-5, 5]$. Pour ce faire, on calculera toutes les valeurs prises par la fonction pour chaque valeur de t allant de -5 à 5, tous les 0.25, et on en déterminera le minimum et le maximum.
2. Écrivez une fonction `AfficherLigne` qui prend en paramètre un entier n et affiche une ligne comportant n caractères *, et termine par un retour à la ligne. Reprenez les exercices 1.1 et 1.2 du TD4 et simplifiez-les en utilisant cette fonction. Pour mémoire, le premier demandait d'afficher un rectangle de largeur et hauteur données. Et le deuxième demandait d'afficher un triangle pointant vers la droite, de hauteur maximale donnée.
3. Écrivez une fonction `RésoudreQuad` qui résout une équation du second degré de la forme $ax^2 + bx + c = 0$: elle prendra en paramètre les valeurs de a, b et c et renverra le résultat sous la forme d'un tableau qui contiendra les solutions (2 en général, 1 si la solution est double et 0 s'il n'y a pas de solution), ainsi que le nombre de solutions. On supposera que la fonction `RacineCarrée` est disponible et renvoie la racine carrée d'un nombre (revoir la méthode de Héron pour avoir une idée de comment ça peut se calculer).

3 Fonctions récursives

Une fonction peut s'appeler elle-même : c'est ce qu'on appelle une fonction récursive !

Un exemple classique est le calcul de la factorielle d'un entier. Pour mémoire, la factorielle d'un entier n est formée par le produit de tous les nombres de 1 à n. Elle est notée $n!$. Par exemple : $1! = 1$, $2! = 2 \times 1 = 2$, $3! = 3 \times 2 \times 1 = 6$, ... , $10! = 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 3628800$.

On peut facilement écrire une fonction `Fact` qui calcule la factorielle d'un nombre n donné en entrée (elle renvoie 1 si $n \leq 0$) :

Fonction *Fact*
  Entrée : n : entier `// La sortie sera` $n!$ `si` $n > 1$`, et 1 sinon`
  Sortie : f : entier `// Contient` $n!$
  Variables locales: i : entier `// Variable de boucle`
  Début
    $f \leftarrow 1$
    Pour i **allant de** *2* **à** n **faire**
      $f \leftarrow i * f$
    FinPour
    Renvoyer f
  Fin

Mais on peut aussi remarquer que, quelque soit n, on a $n! = n \times (n-1)!$. C'est une définition dite récursive car on définit la factorielle de n avec la factorielle de $n-1$. Dit en langage informatique, on a `Fact` (n) = n * `Fact` (n-1).

Si je calcule ainsi 2!, je vais appeler

- `Fact` (2) = 2 * `Fact` (1), ce qui fait appel à la fonction `Fact` (1)
- Or `Fact` (1) = 1 * `Fact` (0), ce qui fait appel à la fonction `Fact` (0)
- Or `Fact` (0) = 0 * `Fact` (-1)... Non, c'est faux !

Il faut donc s'arrêter et une meilleure définition récursive est de dire que `Fact` (n) = n*`Fact` (n-1) **si n est strictement supérieur à 1** et sinon `Fact` (n) = 1. Dès lors le calcul de 2! se fait ainsi :

— `Fact` (2) = 2*`Fact` (1)

— `Fact` (1) = 1 (car 1 n'est pas strictement supérieur à 1). Ici aucun nouvel appel à `Fact` n'est effectué. La fonction s'arrête donc.

On peut alors donner une définition de la fonction `Fact` en pseudo-langage, sous forme récursive, et ainsi sans boucle :

Fonction *Fact*
| **Entrée** : n : entier `// La sortie sera` $n!$ `si` $n > 1$`, et 1 sinon`
| **Sortie** : f : entier `// Contient` $n!$
| **Début**
| | **Si** $n > 1$ **alors**
| | | $f \leftarrow n$ * `Fact` $(n-1)$
| | **Sinon**
| | | $f \leftarrow 1$
| | **FinSi**
| | **Renvoyer** f
| **Fin**

Une version plus courte sera :

Fonction *Fact*
| **Entrée** : n : entier `// La sortie sera` $n!$ `si` $n > 1$`, et 1 sinon`
| **Sortie** : entier `// Contient` $n!$
| **Début**
| | **Si** $n \leq 1$ **alors**
| | | **Renvoyer** 1
| | **FinSi**
| | **Renvoyer** n * `Fact` $(n-1)$
| **Fin**

On remarque ici que la variable f est devenue inutile et on exploite le fait que la fonction se termine dès lors qu'on exécute une instruction **Renvoyer**.

Mais sauriez-vous me dire pourquoi la version suivante ne marche pas ?

Fonction *Fact*
| **Entrée** : n : entier `// La sortie sera` $n!$ `si` $n > 1$`, et 1 sinon`
| **Sortie** : entier `// Contient` $n!$
| **Début**
| | **Renvoyer** n * `Fact` $(n-1)$
| | **Si** $n \leq 1$ **alors**
| | | **Renvoyer** 1
| | **FinSi**
| **Fin**

4 Exercices

1. Écrivez, sous forme récursive, une fonction `exposant` qui prend entrée un réel x et un entier n et renvoie x^n si $n > 0$, et 1 sinon. On remarquera que $x^n = x \times x^{n-1}$.
2. La suite de Syracuse pour un entier S est définie de la manière suivante :

$$u_0 = S \tag{1}$$

$$\forall k \geq 0 \quad u_{k+1} = \begin{cases} \dfrac{u_k}{2} & \text{si } u_k \text{ est pair} \\ 3u_k + 1 & \text{si } u_k \text{ est impair} \end{cases} \tag{2}$$

   Écrivez une fonction récursive qui renvoie le terme d'indice n de la suite pour une valeur donnée de S.