Université Assane SECK de Ziguinchor

Unité de Formation et de Recherche des Sciences et Technologies

Département d'Informatique

Les sous algorithmes : procédures & fonctions

Licence 1 en Ingénierie Informatique

Octobre 2021

©Papa Alioune CISSE

Papa-alioune.cisse@univ-zig.sn

Résumé : Parfois quand on doit écrire un algorithme pour la résolution d'un problème complexe, il est nécessaire de décomposer le problème en des sous- problèmes. En outre, il arrive fréquemment qu'une même suite d'instructions doive être exécutée en plusieurs points d'un algorithme. Dans ces situations, pour chaque sous problèmes et pour chaque suite d'instructions à répéter, on peut écrire un sous algorithme qui sera appelé par l'algorithme principal. Un sous algorithme peut être une procédure ou une fonction et est spécifié par un nom, éventuellement une liste de paramètres formels (paramètres déclaratifs) et éventuellement une valeur de retour. Les paramètres formels, s'ils existent, ils spécifient des données et/ou des résultats du sous problème à résoudre.

1 - DÉFINITIONS ET EXEMPLES

Un sous algorithme permet, entre autres, de regrouper des instructions ou traitements qui doivent être faits de manière répétitive au sein d'un algorithme principal. Par exemple, dans un algorithme traitant des tableaux, on voudrait afficher plusieurs fois des tableaux dont les variables stockent des valeurs différentes. Cependant, même si les valeurs sont différentes, l'affichage d'un tableau se résume toujours à un parcours de toutes les variables utilisées avec une boucle POUR (de l'indice *1* jusqu'à l'indice *taille_du_tableau*), avec un affichage de chaque valeur grâce à l'instruction ***Écrie ( )***.

Il serait utile de regrouper ces instructions d'affichage, pour n'en avoir qu'un seul exemplaire, et de pouvoir s'en servir lorsqu'on en a besoin, sans avoir à saisir les lignes dans le programme. C'est à cela que sert un sous algorithme: il regroupe des instructions auxquelles on peut faire appel en cas de besoin. Il s'agit en réalité d'un petit algorithme qui sera utilisé par l'algorithme principal.

Ces sous algorithmes fonctionnent en « boîte noire » vis-à-vis des autres sous algorithmes ou de l'algorithme principal. Autrement dit, ils n'en connaissent que les entrées (valeurs qui sont fournies au sous algorithme et sur lesquelles son traitement va porter) et les sorties éventuelles (valeurs fournies par les sous algorithmes et qui peuvent être récupérées par les autres sou algorithmes ou par l'algorithme principal).

Comme de nombreuses entités en informatique, un sous algorithme doit être défini avant d'être utilisé, c'est-à-dire que l'on doit indiquer quelles sont les instructions qui la composent : il s'agit de la définition du sous algorithme où on lui associe des instructions. On doit donc trouver une définition du sous algorithme, qui comporte une identification ou ***en-tête*** du sous algorithme, suivie de ses instructions, appelées aussi ***corps du sous algorithme***.

En résumé, un sous algorithme doit être définie et comporter un en-tête pour l'identifier et un corps contenant ses instructions pour la définir.

Un sous algorithme peut être écrit sous forme de fonction ou de procédure. La différence principale entre une procédure et une fonction est qu'une fonction retourne une valeur après avoir effectué des traitement alors qu'une procédure n'en retourne pas. Le choix de l'un ou l'autre dépend du sous problème à traiter. En effet, quand on veut écrire un sous algorithme qui calcule la racine carrée d'un nombre par exemple, il faut nécessairement retourner la valeur issue des traitements du sous algorithme : une fonction est donc appropriée. Cependant, pour visualiser un tableau par exemple, on a juste besoin d'afficher les éléments du tableau sans retourner une valeur quelconque : une procédure est appropriée.

2 - LES PROCÉDURES

2.1 - Déclaration d'une procédure

Une procédure a la forme générale suivante :

```
Procédure nom_de_la_procedure (Liste des paramètres formels avec leurs type séparés par des « , »)
            Déclaration des variables locales de la procédure
            Début
              Suite des instructions de la procédure
            Fin
```

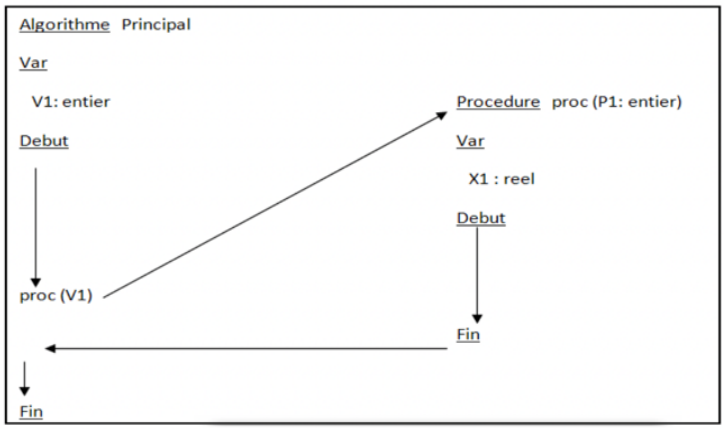
2.2 - Exemple

Une procédure pour le *calcul et l'affichage* de la somme des n premiers entiers.

```
Procédure Somme (n : Entier)        En-tête
            Var i, S : Entier
            Début
              S ← 0
    Corps     Pour i de 1 à n faire
                S ← S + i
              Fin pour
              Écrire ('La somme des ', n, ' premiers entiers est : ', S)
            Fin
```

2.3 - Schéma d'exécution d'une procédure dans un algorithme principal

L'appel d'une procédure se fait en utilisant son nom suivi de ses paramètres séparés par des virgules et entourés par des parenthèses. Quand on appelle une procédure, le contrôle se trouve automatiquement transféré au début de la procédure. Quand toutes les instructions de la procédure ont été exécutées, le contrôle retourne à l'instruction qui suit immédiatement l'instruction d'appel de la procédure.



2.4 - Paramètres formels d'une procédure

Il existe 3 types de paramètres dans une procédure :

- **Paramètres de type « donnée ».** Ils ne changent pas de valeur dans la procédure. Leurs valeurs sont plutôt utilisées seulement sans être modifiées.
- **Paramètres de type « résultat ».** Ils n'ont une signification qu'après l'exécution du sous-programme. Le sous-programme doit déterminer *leurs valeurs.*
- **Paramètres de type « donnée-résultat ».** Ils ont une valeur avant l'exécution du sous-programme et qui peut changer après l'exécution.

2.5 - Modes de passage des paramètres formels

Il existe deux modes de passage des paramètres :

- **Passage par valeur qui concerne les paramètres de type « donnée ».** Le paramètre formel représente une variable locale au sous-programme et tout changement de valeur n'a pas d'effet sur le paramètre effectif. A l'appel, les valeurs des paramètres effectifs sont récupérés dans les paramètres formels sur lesquels le traitement s'effectue.

  Dans la déclaration suivante : ***procédure test (p1 : type1, p2 : type2)***, les paramètres ***p1*** et ***p2*** sont passés par valeur.

- **Passage par adresse qui concerne les paramètres de types « résultat » et « donnée-résultat ».** Le principe consiste à associer aux paramètres formels l'adresse des paramètres effectifs. Toute modification apportée aux paramètres formels affecte les paramètres effectifs.

  Dans la déclaration suivante : ***procédure test (p1 : type1, var p2 : type2, var p3 : type3)***, les paramètres ***p2*** et ***p3*** sont passés par adresse.

  Remarque : Les paramètres passés par adresse sont précédés du mot clé « ***var*** ».

2.6 - Appel d'une procédure

Dans un algorithme principal, une procédure est appelée dans une instruction en spécifiant son nom et ses paramètres qui deviennent des paramètres effectifs (ou paramètres réels) avec le mode de passage respectif. Ainsi, si une procédure est déclarée avec l'entête suivante :

procédure test (p1 : type1, var p2 : type2, p3 : type3, var p4 : type4)

Alors, au moment de l'appel avec les paramètres effectifs q1 et q3 passés par valeur et les paramètres q2 et q4 passés par adresse, l'action prend la forme suivante :

test (q1, &q2, q3, &q4).

Remarque : Les paramètres formels et les paramètres effectifs peuvent porter le même nom.

Exemple 1 : Écrire une procédure pour le calcul et l'affichage de la somme des n premiers entiers. Ici « n » est un paramètre de type « donnée ».

```
Procédure Somme (n : Entier)
            Var i, S : Entier
            Début
              S ← 0
              Pour i de 1 à n faire
                S ← S + i
              Fin pour
              Écrire ('La somme des ', n, ' premiers entiers est : ', S)
            Fin
```

Un algorithme principal qui fait appel à cette procédure effectue la saisie d'une valeur pour ***n*** et fait appel à la procédure dans une action en spécifiant le nom de la procédure et ***n*** qui devient un paramètre effectif.

```
Algorithme calculSommeNpremiersEntier
            Var n : Entier
            Début
              Écrire ('Donner un entier')
              Lire (n)
              Somme (n)
            Fin
```

Exemple 2 : Écriture d'une procédure pour le calcul de la factoriel d'un entier positif. Dans cette procédure, puisque l'affichage de la factoriel n'est pas effectué, il faut donc trouver un moyen de récupérer le résultat. La solution consiste à ajouter à la procédure un paramètre de type « résultat ». Il s'agit ici du paramètre « **f** ».

```
Procédure factoriel (n : Entier, var f : Entier)
            Var i : Entier
            Début
              f ← 1
              i ← 1
              Tant que (i <= n) faire
                f ← f * i
                i ← i + 1
              Fin Tant que
            Fin
```

Un algorithme principal qui fait appel à cette procédure pour récupérer et afficher la factorielle d'un nombre entier saisi au clavier peut être le suivant :

```
Algorithme factoriel_principal
            Var n, fact : Entier
            Début
              Répéter
                Écrire ('Donner un entier')
                Lire (n)
              Jusqu'à (n >= 0)
              factoriel (n, fact)
              Écrire ('La factoriel de ',n, ' est : ', fact)
            Fin
```

Exemple 3 : Écrire une procédure qui permet d'échanger le contenu de deux variables entières a et b reçu en paramètre. Dans cette procédure, a et b sont des paramètres de type « donnée-résultat ». En effet, ils ont des valeurs avant la procédure qui doivent être échangées (donc modifiées) par la procédure.

```
Procédure Échange (var a : Entier, var b : Entier)
            Var tmp : Entier
            Début
              tmp ← a
              a ← b
              b ← tmp
            Fin
```

Un algorithme principal qui fait appel à cette procédure peut être le suivant :

```
Algorithme échange_principal
            Var x, y : Entier
            Début
              Écrire ('Donner la valeur de x')
              Lire (x)
              Écrire ('Donner la valeur de y')
              Lire (y)
              Écrire ('Avant échange : x = ',x, ' et y = ', y)
              Échange (x, y)
              Écrire ('Après échange : x = ',x, ' et y = ', y)
            Fin
```

2.7 - Les variables locales et globales

➢ Variables locales

Une variable est dite locale si elle est définie dans la partie de déclaration des variables propres à la procédure. Elle n'est accessible que dans la procédure où elle a été définie. Dans « ***Exemple 3*** », ***tmp*** est une variable locale à la procédure ***Échange***.

➢ Variables globales

Une variable est dite globale si elle est définie au niveau de l'algorithme principal qui appelle la procédure, c'est à dire une variable utilisée par la procédure et qui n'est pas déclarée dans cette procédure : on dit qu'elle est visible par l'algorithme principal et par toutes les autres procédures utilisées. L'échange d'information entre les procédures et l'algorithme peut se faire à travers les variables globales.

Dans l'exemple qui suit, nous avons une variable globale qui s'appelle ***nb_population*** (représentant le nombre total de personnes d'une population donnée), une procédure ***naissance***

(n : Entier) pour enregistrer les naissances dans cette population à chaque fois qu'une femme accouche (n étant le nombre d'enfants nés) et une procédure ***décès ( )*** pour enregistrer les décès dans cette population.

```
Algorithme population
            Var nb_population : Entier {déclaration de la variable globale }

            Procédure naissance (n : Entier)
              Début
                  Nb_population ← nb_population + n
              Fin naissance

            Procédure décès ( )
              Début
                  Nb_population ← nb_population - 1
              Fin décès

            Début
              Écrire ('Donner la population initiale')
              Lire (nb_population)              {si on donne ici 50 par exemple}
              naissance (2)
              naissance (1)
              naissance (2)
              décès ()
              naissance (2)
              décès ()
              Écrire (La population actuelle est : ',nb_population) {on aura alors ici 55}
            Fin
```

3 - LES FONCTIONS

Une fonction est un sous-programme qui retourne un résultat unique de type scalaire (entier, réel, caractère, booléen). ***Il s'agit en fait de la première différence entre une procédure et une fonction. Cela veut dire aussi que tout ce qui est dit sur les procédures est aussi valable sur les fonction.***

3.1 - Déclaration d'une fonction

Une fonction a la forme générale suivante :

```
Fonction nom (Liste des paramètres avec leurs type séparés par des « , ») : type_résultat
            Déclaration des variables locales de la fonction
            Début
              Suite des instructions de la fonction
            Fin
```

3.2 - Appel d'une fonction

Comme une fonction rend un résultat unique, son appel s'effectue dans un algorithme principal en affectant le résultat retourné à une variable déclarée dans cet algorithme. ***Il s'agit ici de la seconde et dernière différence avec une procédure***.

L'appel prend la forme suivante :

identificateur ← nom_fonction (listeParamètresEffectifs)

Remarque : Les types de la variable identifiée et du résultat retourné doivent être compatibles.

Exemple 4 : Écrire une fonction pour le calcul de la somme des n premiers entiers.

```
Fonction Somme (n : Entier)
            Var i, S : Entier
            Début
              S ← 0
              Pour i de 1 à n faire
                S ← S + i
              Fin pour
              Somme ← S
            Fin
```

Un algorithme principal qui fait appel à cette fonction effectue la saisie d'une valeur pour n et fait appel à la fonction dans une action qui affecte le résultat qu'elle retourne à une variable déclarée dans l'algorithme:

```
Algorithme Somme_principal
            Var n, S : Entier
            Début
              Écrire ('Donner la valeur pour n')
              Lire (n)
              S ← Somme (n)
              Écrire ('La somme des ',n, ' premiers entiers est ', S)
            Fin
```

4 - Les fonctions récursives

4.1 - Définition récursive

Une récursivité est définie en fonction d'un objet ou d'une action de même nature mais de complexité moindre. Par exemple, on peut définir le factoriel d'un nombre « n » positif de deux manières:

- définition non récursive (ou itérative): n ! = n * n-1 * ... 2 * 1
- définition récursive :

$$\begin{cases} n! = n * (n-1)! \\ 0! = 1 \end{cases}$$

On remarque qu'une définition récursive est composée de deux parties: une partie strictement récursive et une partie non récursive (base) servant de point de départ à l'utilisation de la définition récursive.

4.2 - Fonction récursive

Les fonctions récursives sont des fonctions qui sont appelées depuis leur propre corps de fonction, directement ou indirectement, à travers éventuellement une ou plusieurs fonctions relais. Si la fonction P appelle directement P, on dit que la récursivité est directe. Si P appelle une fonction P1, qui appelle une fonction P2 et qui enfin appelle P, on dit qu'il s'agit d'une récursivité indirecte.

Si la fonction P appelle dans son propre corps la fonction P, il est logique de penser que l'exécution ne s'arrêtera jamais. Il est donc primordial qu'une des branches de la fonction P permette de stopper la récursivité.

Exemple : On veut calculer la somme des n premiers entiers positifs. Une définition récursive de cette somme serait:

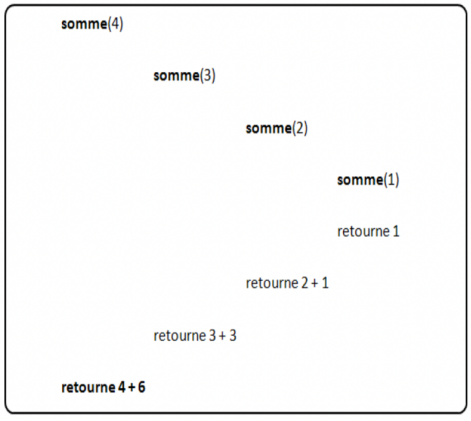
$$\begin{cases} somme\ (n) = n + somme\ (n-1)! \\ somme\ (1) = 1 \end{cases}$$

Ce qui se traduit par la fonction récursive suivantes :

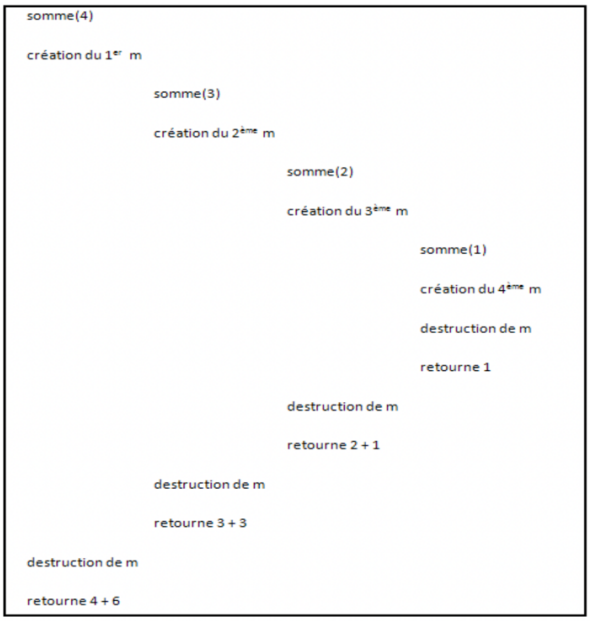
```
Fonction somme(n : entier) : entier
            Var m : entier
            début
            si (n = 1) alors
              somme ← 1
            sinon
              début
              m ← somme (n-1)
              somme ← n + m
            fin sinon
      Fin
```

4.3 - Exécution d'une fonction récursive

Supposant qu'on appelle la fonction somme (4). Puisque 4 ≠ 1, cette fonction va s'appeler elle-même avec la valeur 3 (somme (3)). Ce nouvel appel se termine en renvoyant une valeur (dans cet exemple la valeur retournée est 6 = 3 + 2 + 1). Cette valeur sera ajoutée à la valeur 4 et l'appel de somme (4) retournera la valeur 10 (4 + 6). L'appel somme (3) suivra le même processus et appellera somme (2) qui lui-même appellera à somme (1) qui retourne 1.



Lorsqu'une fonction récursive définit des variables locales, un exemplaire de chacune d'entre elles est créé à chaque appel récursif de la fonction. Dans notre exemple ***somme (4)***, la variable locale ***m*** est créée 4 fois. Ces variables sont détruites au fur et à mesure que l'on quitte la fonction comme toute variable locale d'une fonction. Il en est de même des paramètres des fonctions.


somme(4)
création du 1er m
somme(3)
création du 2ème m
somme(2)
création du 3ème m
somme(1)
création du 4ème m
destruction de m
retourne 1
destruction de m
retourne 2 + 1
destruction de m
retourne 3 + 3
destruction de m
retourne 4 + 6