Université A. Mira Béjaia
Faculté de Sciences Exactes
Département Mathématiques
L1 Math (tronc commun)
2025/2026

Module : Algorithmique et Structures de Données 1 (ASD1)

Chapitre 4 : Les actions paramétraient (Les sous Programmes)

1. Introduction

Un **sous-programme** est un **bloc d'instructions indépendant**, conçu pour effectuer une tâche spécifique à l'intérieur d'un programme principal. En programmation, les sous-programmes permettent de **diviser un programme** complexe **en plusieurs parties** plus **petites et réutilisables**, facilitant ainsi la lecture, et la maintenance du programme.

Un sous-programme est une séquence d'instructions nommée et définie une seule fois, mais qui peut être appelée plusieurs fois à différents endroits dans le programme.

Il existe deux types principaux de sous-programmes dans la plupart des langages de programmation :

- **Procédures** : Sous-programmes qui exécutent une série d'instructions sans nécessairement retourner de valeur (en langage C, les fonctions de type **void** jouent ce rôle).
- **Fonctions** : Sous-programmes qui **retournent une valeur** après leur exécution.

2. Les procédures

Une procédure est un ensemble d'instructions regroupées sous un **nom**, qui réalise un **traitement particulier** dans un programme **lorsqu'on l'appelle**. Une procédure (sous-programme en général) possède **un nom, des variables**, des instructions, un **début et une fin**. Mais elle ne peut pas s'exécuter indépendamment d'un programme principal.

2.1. Syntaxe de déclaration d'une procédure

```
Procedure Nom_Procedure (les paramétres de la procedure ) ;

Const ……..        Déclaration des constantes et
Var ……            variables locales

Debut

    Les instructions de la procedure

Fin ;
```

2.2. Syntaxe d'appel d'une procédure

nom_procedure (les paramètres séparés par des virgules) ;

2.3. La structure d'un algorithme utilisant un sous programme (une procédure)

```
Algorithme nom_algorithme ;
        Const ;
        Var ;
        Struct ;
        Procédure nom_procedure (liste de paramètres) ;
                const          | section
                var            |
                .              | de déclaration
                .              |
        debut
        |     Description des actions effectuées par la procédure.
        |fin ;
début
|       action1 ;
|       action2 ;
|       .......
|       nom_procedure (liste de paramètres séparés par des virgules) ;
|       .......
|       actionn ;
|
|fin.
```

Exemple1 :

```
Algorithme exemple1 ;
Var x, y, Som : entier;
Procédure Somme (a :entier ; b :entier ; var S :entier) ;
  Debut
    S←a+b ;
  Fin ;
Début { L'algorithme principal }
| lire(x,y) ;
| Somme(x, y, Som) ;
| ecrire(la somme=', Som) ;
|Fin.
```

3. Les variables globales et les variables locales

Dans les sous-programmes (fonctions et procédures), les variables peuvent être **locales** ou **globales**, en fonction de leur portée et de leur accessibilité dans le programme. Voici les différences et les utilisations des deux types :

3.1. Les variables Locales

Les **variables locales** sont définies à l'intérieur d'un sous-programme (fonction ou procédure) et ne sont accessibles qu'à l'intérieur de ce sous-programme. Elles sont créées lorsque le sous-programme est appelé et sont supprimées une fois le sous-programme terminé.

❖ **Les caractéristiques des variables locales :**

- **Portée limitée** : ne sont accessibles qu'à l'intérieur du sous-programme où elles sont définies.
- **Indépendance** : les mêmes noms de variables locales peuvent être utilisés dans différents sous-programmes sans conflit.
- **La création** : elles existent uniquement pendant l'exécution du sous-programme.

3.2. Les variables Globales

Les **variables globales** sont définies en dehors de tous les sous-programmes, dans la partie déclaration de l'algorithme principale, et sont accessibles par tous les sous-programmes dans l'algorithme (le programme). Elles sont créées une fois et existent durant l'exécution de l'algorithme (le programme).

❖ **Les caractéristiques des variables globales :**

- **Portée étendue** : les variables globales sont accessibles dans tous les sous-programmes de l'algorithme (du programme).
- **Partage de données** : sont utiles pour partager des valeurs entre différents sous-programmes.
- **La création** : occupent de la mémoire pendant toute l'exécution du programme.

4. Les paramètres des sous programmes

Les sous-programmes communiquent entre eux à travers des paramètres.

4.1. Les paramètres formels et les paramètres effectifs

4.1.1. Paramètre formel

Les variable utilisée dans le corps du sous-programme (il fait partie de la description du sous-programme).

4.1.2. Paramètre effectif (réel)

Il s'agit de la variable fournie lors de l'appel du sous-programme (valeurs fournies pour utiliser la procédure).

- **Remarque :** les paramètres effectifs du programme principal et les paramètres formels du sous-programme doivent avoir le même nombre de variables ainsi que le même type. Ainsi, il est préférable d'utiliser des noms différents pour les paramètres formels et les paramètres effectifs.
- **Remarque :** Une procédure (ou fonction) sans paramètres doit être toujours suivie des parenthèses vides **nom_procedure** ().

Exemple 2 :

```
Algorithme exemple2 ;
Var x, double : entier;
Procédure calcul_double (a :entier ; var d :entier) ;
Debut
| d←a*2 ;                          Paramètre formel
|Fin ;
Début
| x←3 ;          Paramètre formel
| calcul_double (x, double) ;
| ecrire(la valeur de double=', double) ;
|Fin.
```

4.2. Le passage des paramètres

4.2.1. Le passage par Valeur

Lorsque l'on passe un argument **par valeur**, on transmet une copie de la valeur de cet argument au sous-programme. Ainsi, toute modification effectuée dans le sous-programme n'affecte pas la valeur originale de la variable dans le programme principal.

- **Caractéristiques :**
  - Une copie de la variable est créée.
  - Toute modification dans le sous-programme n'affecte pas la variable d'origine.
  - Utile quand on souhaite protéger les variables d'origine contre les modifications.

- **La syntaxe :**

Procedure nom_procedure (param1 : type1 ; param2 : type2 ;…; paramn: typen);

4.2.2. Le passage par Adresse (ou par Référence)

Lorsqu'on passe un argument **par adresse**, on transmet l'adresse (ou le pointeur) de la variable au sous-programme. Cela signifie que toute modification de cette variable dans le sous-programme affectera la variable originale, car le sous-programme agit directement sur la mémoire où la variable est stockée.

- **Caractéristiques :**

  - Le sous-programme accède directement à l'adresse mémoire de la variable.
  - Toute modification affecte la variable originale.
  - Utile quand on souhaite que le sous-programme modifie directement les valeurs originales.

- **La syntaxe :**

Procedure nom_procedure (**var** param1 : type1 ; **var** param2 : type2 ;…;**var** paramn: typen);

Exemple 3
Permutation de deux valeurs avec le passage **par valeur**, et avec le **passage par adresse**

- **Passage par valeur :**

```
Algorithme exemple ;

        Var     x, y : entier ;
        Procédure  permute (a : entier ; b : entier) ;
              var   c : entier ;
        debut
        |     c←a ;
        |     a←b ;
        |     b←c ;
        |fin ;
début
|       x←2;
|       y←6 ;
|       permute (x, y) ;
|       ecrire ('valeur de x et y =', x, y) ;
|
|fin.
```

Le résultat sera : valeur de x et y = 2 6

➢ **Passage per adresse :**

```
Algorithme exemple ;

        Var    x, y : entier ;
        Procédure  permute (var  a : entier ; var b : entier) ;
              var   c : entier ;
        debut
        |     c←a ;
        |     a←b ;
        |     b←c ;
        fin ;
début
|       x←2;
|       y←6 ;
|       permute (x, y) ;
|       ecrire ('valeur de x et y =', x, y) ;
|
fin.
```

Le résultat sera : valeur de x et y = 6 2

5. Les fonctions

Une fonction est **procédure particulière** qui retourne un et **un seul résultat**, au programme appelant, **de type simple** : entier, réel, booléen ou caractère.

L'appel des fonctions est différent de l'appel des procédures, l'appel d'une fonction doit obligatoirement se trouver à l'intérieur d'une instruction (affichage, affectation, …) qui utilise sa valeur retournée.

5.1. Syntaxe de déclaration d'une fonction

fonction Nom_fonction (les paramétres de la procedure) : **type de résulat ;**

Const ……..
Var …… Déclaration des constantes et **variables locales**

Debut

Les instructions de la fonction

Nom_fonction← valeur ; // ou **retourner (valeur).**
cette instruction spécifie la valeur de retour d'une fonction, la valeur de fonction est calculée et affectée.
Fin ;

5.2. Syntaxe d'appel d'une fonction

variable←nom_fonction(liste des paramètres) ;// dans le cas d'une instruction d'affectation.

Ecrire (nom_fonction(liste des paramètres)) ; // dans le cas d'une instruction d'affichage.

Exemple : Ecrire une fonction qui calcul le factoriel d'un nombre passé en paramètre

```
Algorithme factorielle ;

        Var    nbre : entier ;
        fonction  fact (n : entier) : entier ;
            var   i, f : entier ;
        debut
        |     f←1 ;|
        |     pour i allant de 2 jusqu'à n faire
        |     |  f←f*i ;
        |     |finpour ;
        |
        |    fact←f ; (transmission du résultat)
        |fin ;
début
|       lire (nbre) ;
|       ecrire (la factorielle du nombre =', fact (nbre)) ;
|
|fin.
```

On n'a pas besoin de lecture du nombre dans la fonction puisqu'on a passé sa valeur comme paramètre.

6. Les fonctions/ les procédures récursives

Une fonction récursive est une fonction qui **s'appelle elle-même** directement ou indirectement pour résoudre un problème en le divisant en **sous-problèmes plus simples**. La récursivité repose sur **deux principes** fondamentaux :

1. **Cas de base** (Condition d'arrêt): C'est la condition qui met fin à la récursion. Elle correspond à un cas où la solution est simple et ne nécessite plus d'appels récursifs.
2. **Réduction vers le cas de base** : À chaque appel récursif, le problème est réduit de manière à se rapprocher du cas de base.

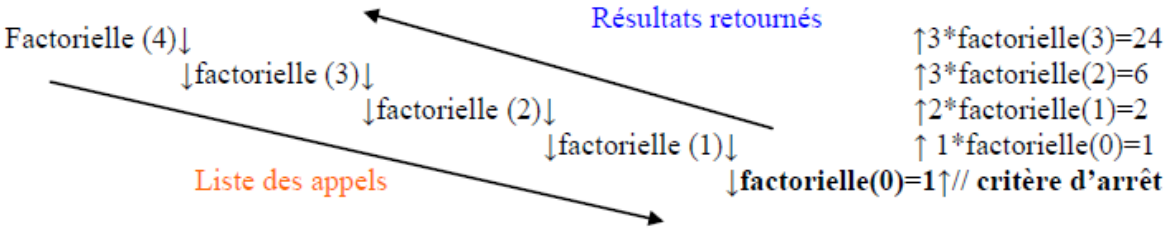
➢ **Caractéristiques d'une fonction Récursive**

- **La structure répétitive** : Les problèmes résolus par une fonction récursive présentent une structure auto-similaire (le problème peut être divisé en sous-problèmes identiques ou similaires), sans utilisation d'une structure itérative (boucle).
- **Les conditions d'arrêt** : Une fonction récursive doit avoir une ou plusieurs conditions d'arrêt (cas de base) pour éviter une récession infinie.

Exemple : une fonction récursive qui calcule le factoriel d'un nombre n passé en paramètre ;

```
Fonction factorielle (n : entier) : entier ;
Debut
|     si   (n=0)  alors      factorielle ← 1 ;
|     |sinon
|     |       factorielle ← n * factorielle (n-1) ;
|     |finsi ;
|fin ;
```

Liste des appels : n=4



Exercice :

Ecrire une **fonction récursive** qui calcule la puissance a^b, tel que ***a, b*** deux nombres entiers positifs passés en paramètres.

7. Les avantages des sous-programmes

- **La réutilisabilité** : Ils peuvent être appelés plusieurs fois depuis différentes parties du programme.
- **La modularité** : Ils aident à diviser le programme en modules ou tâches spécifiques.
- **La clarté** : Ils améliorent la lisibilité du code en organisant les tâches par fonctionnalité.
- **La facilité de maintenance** : Les sous-programmes peuvent être modifiés sans impacter d'autres parties du programme.