Ministère de l'enseignement supérieur et de la recherche scientifique
Université Djillali BOUNAAMA - Khemis Miliana (UDBKM)
Faculté des Sciences et de la Technologie
Département de Mathématiques et d'Informatique

Chapitre 1

Les sous-programmes : Fonctions et Procédures

MI-L1-UEF221 : Algorithmiques et Structures de Données II

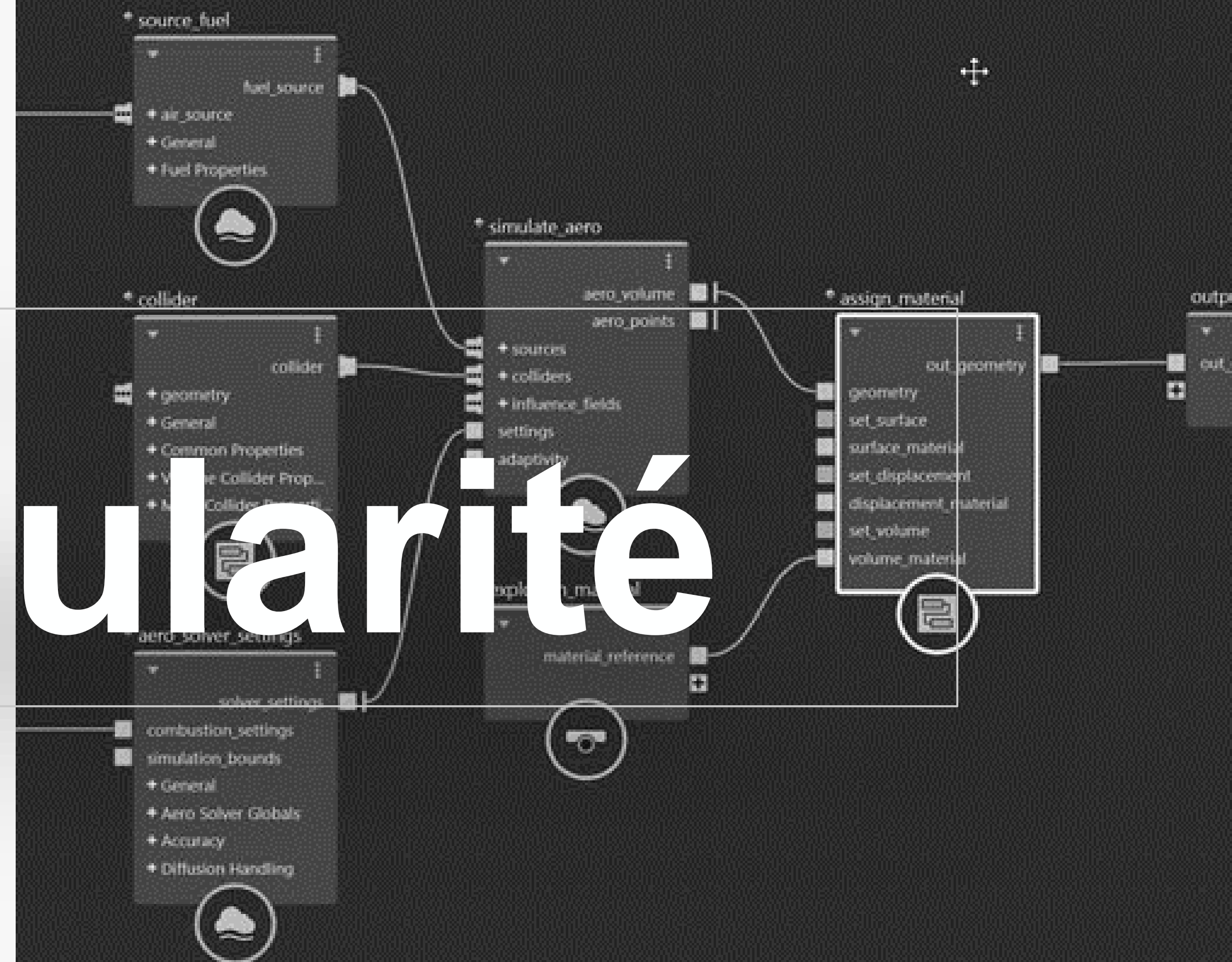
Noureddine AZZOUZA

n.azzouza@univ-dbkm.dz

Plan du Cours

1. La Modularité
2. Le passage des paramètres
3. Les variables locaux et les variables globaux
4. Les Fonctions
5. Les Procédures

La Modularité



La Modularité

Problème

Calcul de Combinaison

Trouver le nombre de combinaison de p objets parmi n tel que $C_n^p = \frac{n!}{p!*(n-p)!}$

```
Algorithme Calcul_Combinaison ;
Var n,p,c : entier ;
      fact_n, fact_p, fact_np: entier;
Debut
   //les entrées
   Lire (n,p);

    //manipulation des données
    fact_n = 1;
   Pour i ← 1 à n Faire
         fact_n ← fact_n * i;

    fact_p = 1;
   Pour i ← 1 à p Faire
         fact_p ← fact_p * i;

    fact_np = 1;
   Pour i ← 1 à (n-p) Faire
         fact_np ← fact_np * i;

    c ← fact_n/(fact_p*fact_np);

    //les sorties
    Ecrire ('le nombre de combinaisons=',c);
Fin.
```

La Modularité

Problème

Répétition du calcul de factorielle

```
fact_n = 1;
Pour i ← 1 à n Faire
    fact_n ← fact_n * i;
```

```
fact_p = 1;
Pour i ← 1 à p Faire
    fact_p ← fact_p * i;
```

```
fact_np = 1;
Pour i ← 1 à (n-p) Faire
    fact_np ← fact_np * i;
```

Comment écrire la solution une seul fois? Organiser le code ?

Modules / Sous-Programmes

La Modularité

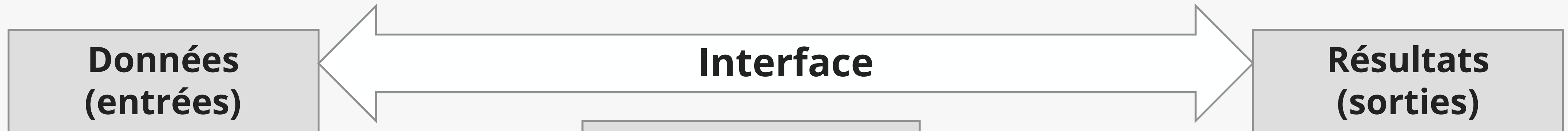
Définition

- ✓ Un Sous-programme ou module est un ensemble d'instructions avec une interface bien définie qui réalise une tache précise.

- ✓ Le but d'un sous-programme est:
  1. Recevoir des données d'entrées
  2. Effectuer un traitement/transformation de ces données
  3. Retourner un ou plusieurs résultats

- ✓ **L'interface** est constituée des entrées / sorties du module. Elle permet d'établir le lien entre le module et son environnement (Algorithme principale, autres modules).

Structure



Type qui dépends de la sortie

Type de Sous Programme

0 à n entrées → **Nom_sous_programme** → **0 à n sorties**

Rôle de Sous Programme

Nom unique et significatif qui est utilisé dans la déclaration et appel

Rôle qui indique ce que fait le sous-programme exactement

Types

✓ Selon le nombre et le type de sorties, on distingues deux (02) types de sous-programmes (modules):

1. **Fonction** : Lorsque le module retourne un seul (1) résultat et que ce résultat est une donnée de type élémentaire (de base), exemple:
   - ➢ Fonction qui calcul la somme d'un tableaux d'entier (retourne ***un entier***)
   - ➢ Fonction qui vérifie si une nombre est premier (retourne ***un booléen***)
2. **Procédure** : Lorsque le module retourne 0 à n résultats ou le résultat est de type structuré, exemples:
   - ➢ Procédure qui affiche une matrice (retourne ***0 résultat***)
   - ➢ Procédure résout une équation du 2ème degrés (retourne ***2 résultats***)
   - ➢ Procédure qui inverse le contenu d'un tableau d'entier (retourne ***un tableau***)

Qualités

✓ Afin de décider si une suite d'instruction mérite d'être conçue sous forme de sous-programme ou module, on doit vérifier les qualités suivantes:

1. **La réutilisation** : un module est conçu pour qu'il soit réutilisé dans plusieurs solutions. Il doit être généraliser au maximum possible.

2. **L'indépendance**: éviter d'utiliser des variable globales dans un module pour qu'il soit indépendant de l'algorithme principal. Même chose pour les lectures et écritures.

3. **La simplicité**: garder votre code lisible et concevoir un module qui répond à une tache précise.

La Modularité

Définition

- ✓ **La Modularité** : « *C'est un mode de pensée visant à construire des algorithmes en partant d'un niveau très général et en détaillant peu à peu chaque traitement, jusqu'à arriver au niveau de description le plus bas* ».

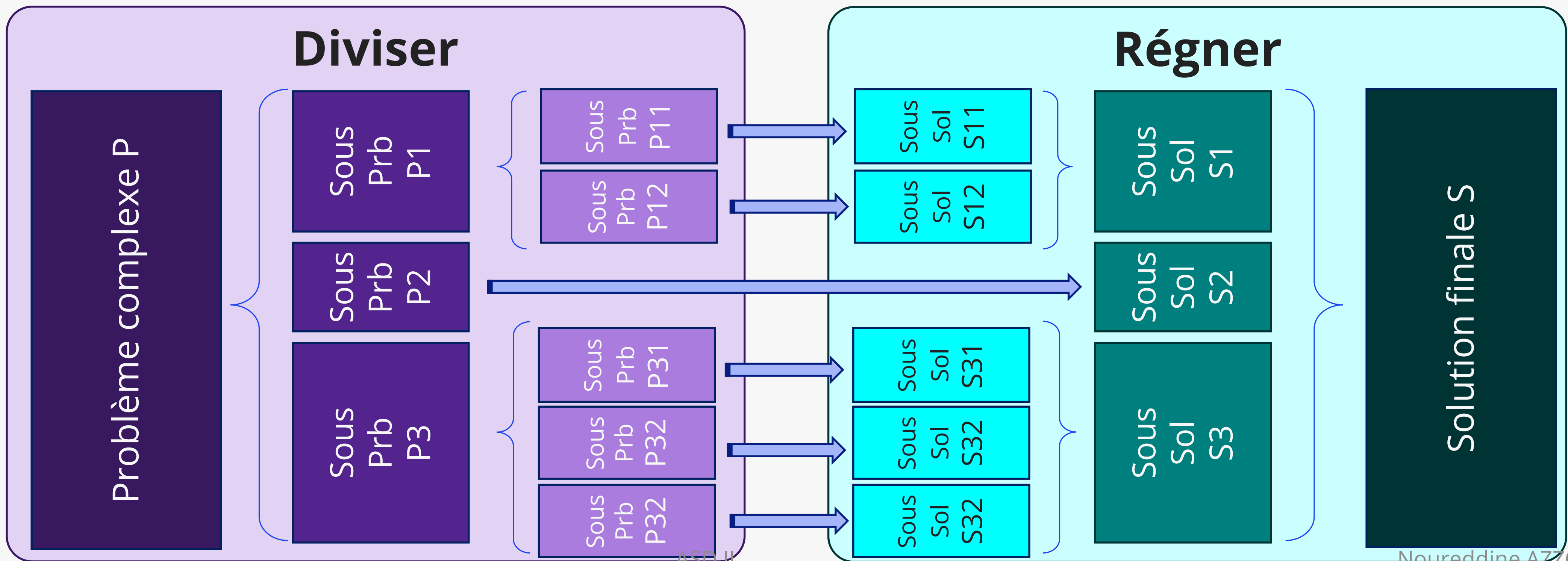
- ✓ **La Modularité** : est une **Analyse Descendante** qui **divise** (découpe) un problème en sous problèmes pour **régner** (combiner) les sous solution et obtenir un résultat global.

- ✓ **La Modularité** : est la base de la **programmation structurée** consiste à résoudre un problème en construisant des **modules** simples, lisibles et réutilisables.

La Modularité

Objectifs

- ✓ Découper (diviser) un problème complexe en sous problèmes simples qui seront résolus séparément.
- ✓ Proposer une solution à un (sous) problème une et une seule fois



Étapes de la Démarche Modulaire

1ère ETAPE : Compréhension du problème

2ème ETAPE : Analyse et Conception

- Découpage Modulaire
- Construction des Modules

3ème ETAPE : Réalisation

Découpage Modulaire

✓ Découpe le problème en modules cohérents:

- ❑ Commencer à extraire les modules évidents qui sont facile à les détecter.
- ❑ Améliorez et enrichissez le découpage au fur et à mesure que vous avancer dans la résolution du problème.

Construction des Modules

- ✓ Pour construire une module, on commence par sa description:
  - ❑ Dessine le Module
  - ❑ Lui donner une nom
  - ❑ Définir son interface
  - ❑ Préciser sa nature (fonction ou procédure)
  - ❑ Indiquer son rôle

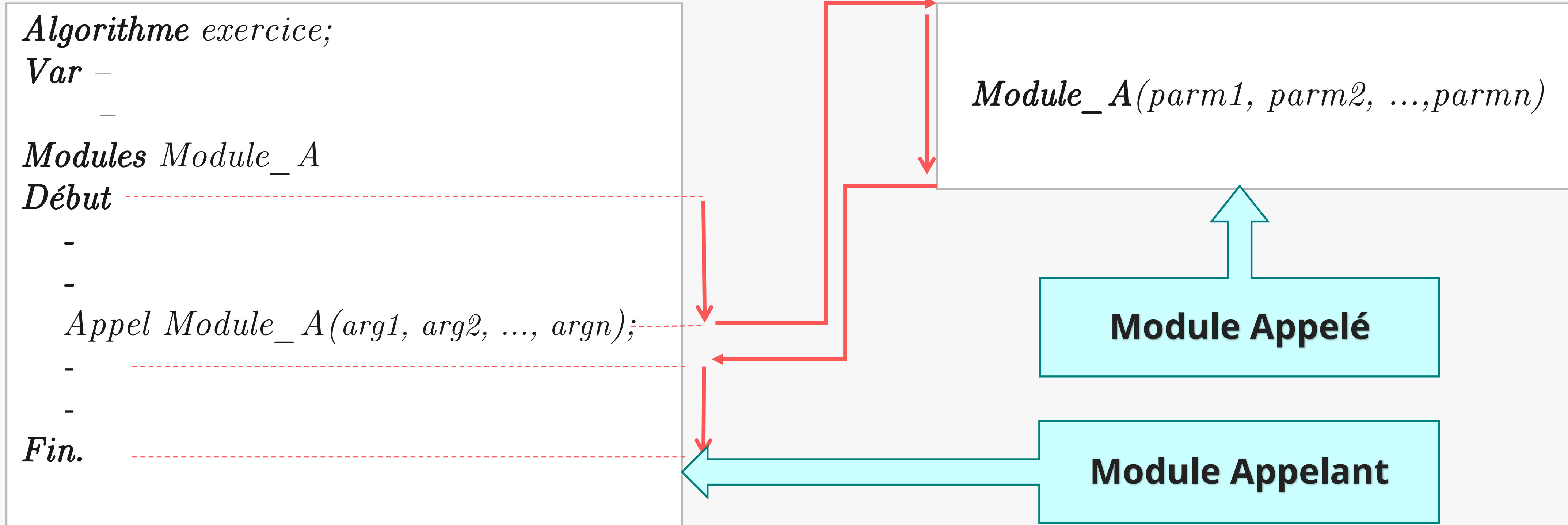
- ✓ Si le module existe déjà, on le construit pas. On donne sa description seulement

Avantages

- ✓ Découpage en modules cohérents se fait par une Approche Descendante
- ✓ Simplifie la conception
- ✓ Construction indépendante et séparée des modules
- ✓ Lisibilité et facilité de compréhension des algorithmes
- ✓ Facilité de maintenance t de mise à jour du code
- ✓ Réutilisation des modules (sous-programmes) déjà conçus

Communication

Communication entre Modules



- ✓ Quand un **appel** à un module est rencontré,
  - ❑ Suspendre l'exécution du ***module appelant*** (Par exemple: Algorithme principale)
  - ❑ Démarrer et exécuter le ***module appelé*** (Par exemple : Module_A)
  - ❑ Reprendre l'exécution du module appelant juste après l'instruction d'appel

Paramètres et Arguments

- ✓ Les variables utilisées lors de la construction du module (en-tête du sous-programme) s'appels ***Paramètres*** ou ***paramètres formels***
  - ❑ **Exemple** : les paramètres (ou paramètre formels) du module « **Module_A** » sont : $parm1,\ parm2,\ \ldots,parmn$
- ✓ Les variables utilisées lors de l'appel (utilisation) d'un module s'appels ***Arguments*** ou ***paramètre effectifs***. Ils replacent les paramètres formels lors de l'appel.
  - ❑ **Exemple** : les arguments (ou paramètre effectifs) du module « **Module_A** » utilisés dans son appel à l'algorithme principale sont : $arg1,\ arg2,\ \ldots,\ argn$
- ✓ Le ***nombre*** d'arguments doit correspondre au nombre de paramètres.
- ✓ L'***ordre*** des arguments doit correspondre à l'ordre des paramètres.
- ✓ Le ***type*** du k^ième^ argument doit correspondre au type du k^ième^ paramètre.
- ✓ La correspondance entre les arguments et les paramètres se fait en utilisant l'**ordre**.

Modes de Passage

Modes de Passage des Paramètres

- ✓ C'est la ***substitution*** des paramètres ***formels*** par les paramètres ***effectifs*** lors de l'appel d'un sous-programme.
- ✓ On distingue deux mode de passage :
  - ❑ ***Passage par Valeur*** : Toute modification du contenu du paramètre dans le programme appelé est sans conséquence sur la valeur du paramètre effectif dans le programme appelant.
  - ❑ ***Passage par Variable (par Référence)*** : toute modification du contenu du paramètre formel entraîne automatiquement la modification du paramètre effectif.
- ✓ Les paramètres formels passé par variable sont précédés par le mot clé **VAR** dans l'entête du sous-programme.

*type_module nom_fonction (Paramètre formels d'entrés; **VAR** Paramètre formels de sorties);*

Passage par Valeur

- ✓ Copier les valeur des arguments au début du sous-programme.
- ✓ C'est comme en fait une affectation des valeurs des arguments dans les paramètres formels associés.
- ✓ Peut recevoir n'importe qu'elle expression (constant, variable, expression, appel de fonct …)

❑ **Exemple** : sous-programme (fonction) qui calcule la surface d'un rectangle.

Appel

$surf \leftarrow Surface\ (x,\ y);$

équivalent à

Module

```
Fonction Surface(long, larg:réel): réel;
Var      S: entier;
Début
    S ← long * larg;
    Surface ← S;
Fin;
```

```
Fonction Surface(long, larg:réel): réel;
Var      S: entier;
Début
    long ← x; larg ← y;
    S ← long * larg;
    Surface ← S;
Fin;
```

Passage par Variable

- ✓ Dans le passage par variable (ou référence) le paramètre devient lui-même l'argument,
- ✓ c'est-à-dire que le paramètre devient un alias de l'argument.
- ✓ Ne peut être lié qu'au variables.
  - **Exemple** : sous-programme (procédure) qui swap (échange) les valeurs de deux variables.

Echange (a, b); — **Appel**

équivalent à

Module

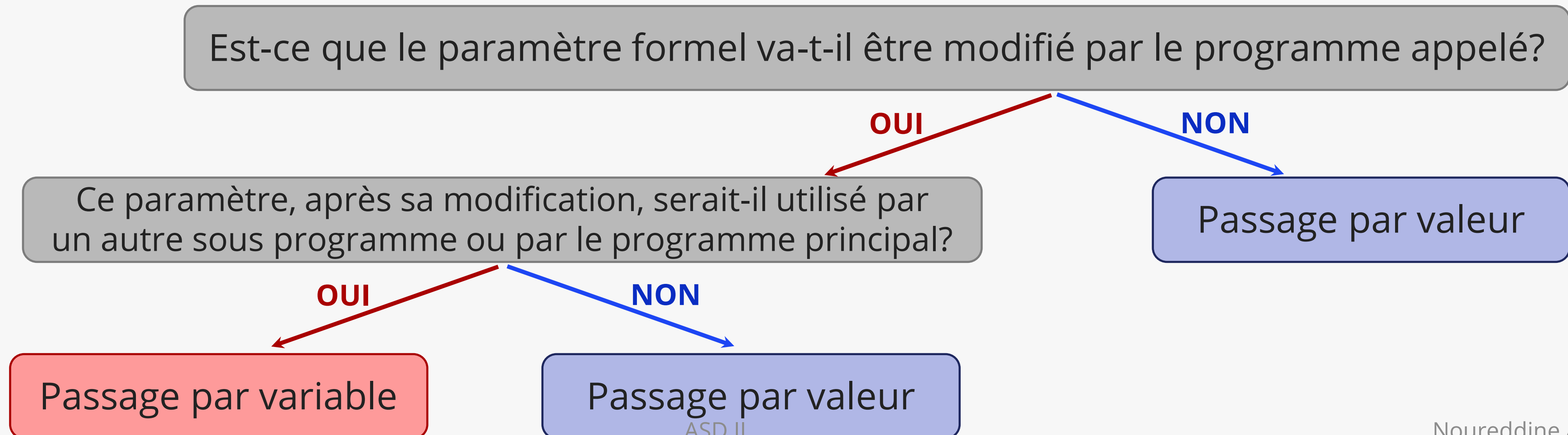
```
Procédure Echange(VAR x, y:entier);
Var      z: entier;
Début
    z ← x;
    x ← y;
    y ← z;
Fin;
```

```
Procédure Echange(VAR x, y:entier);
Var      z: entier;
Début
    z ← a;
    a ← b;
    b ← z;
Fin;
```

Modes de Passage des Paramètres

- ***Passage par Valeur*** : est adopté lorsqu'on souhaite que le module nous restitue **la même valeur** que le paramètre avait à l'entrée, ou le paramètres **n'est pas utilisé** dans d'autres modules (inutile pour trouver le résultat finale).

- ***Passage par Variable (par Référence)*** : est adopté lorsque le paramètre d'entrée est **modifié** lors de l'exécution d'un sous-programme et c'est le **contenu modifié** du paramètre que l'on souhaite.



Remarques

Il est recommandé d'utiliser :

- Un **passage par valeurs** pour les paramètres d'**entrée** d'une **fonction**.

- Un **passage par variable** pour tous les paramètres de **sortie** d'une **procédure**.

Les Variables Locales et les Variables Globales

Il existe deux catégories de variables:

- ***Variables Locales***: qui sont définie dans un module et qui ne peuvent pas être manipulés que dans ce module.
- ***Variables Globales***: qui sont définie dans un module appelant et peuvent être manipulés dans ce module et dans tous les modules appelé par ce module.

- **La portée** : d'un variable est l'ensemble des modules ou ce variable est accessible.

Les Variables Locales et les Variables Globales

```
Module_Appelant
Var     A, B, X: réel;
    Module_1
    Var     X: entier;
            T: booléen;
        Module_2
        Var     C: entier;
    Module_3
    Var     X, Y, Z: entier;
Début
    -
    -
Fin;
```

Variable	Portée
A,B	Module_Appelant , Module_1, Module_2, Module_3
T	Module_1, Module_2
C	Module_2
X:déclaré au Module_Appelant	Module_Appelant
X:déclaré au Module_1	Module_1, Module_2
X:déclaré au Module_3	Module_3
Y, Z	Module_3

Les Fonctions

Définition & Description

✓ Une fonction est sous-programme (module) qui retourne un seul résultat (sortie unique) de type simple (élémentaire) : entier, réel, booléen, caractère. Elle peux recevoir 0 à n paramètres d'entrées.

Description d'une fonction

Structure & Syntaxe

Les Fonctions

Entête

Fonction nom_fonction (Liste des paramètres formels d'entrée :Type): Type de Retour;

Corps

Type –
Const –
Var –

{ **Déclaration des données (objets) locales** }

Début

-
-
-
-

{ **Traitements** }

nom_fonction ← résultat;

Fin;

Propriétés & Remarques

- ✓ Le corps d'une fonction peut contenir toutes les **déclarations** (Type, Const, Var, ..) et les **structures** algorithmiques (Affectation , Répétition, Conditionnel…).
- ✓ Le résultat calculé (valeur de retour) doit être **transmit dans le nom de la fonction**. Cette affectation est situ –dans la plupart des cas- à la fin de la fonction.
- ✓ Les **paramètres formels** décrivent les paramètres d'entrées utilisées dans la fonction ainsi que leur **type** et leur **mode de passage**.
- ✓ Dans les fonctions, les paramètres formels sont utilisés en mode de passage **par valeur.**

Appels

- ✓ L'appel d'une fonction peut être utilisé comme:
  - ❑ ***Expression*** dans une **affectation**,
  - ❑ ***Opérande*** dans une **condition**
  - ❑ ***Argument*** dans un **appel** de **procédure** ou de **fonction**
- ✓ **Exemples:**
  - ❑ X ← Premier (a)
  - ❑ Si Premier (a) = Vrai alors écrire (a, 'est premier')
  - ❑ Res ← Prem (Fact(n))

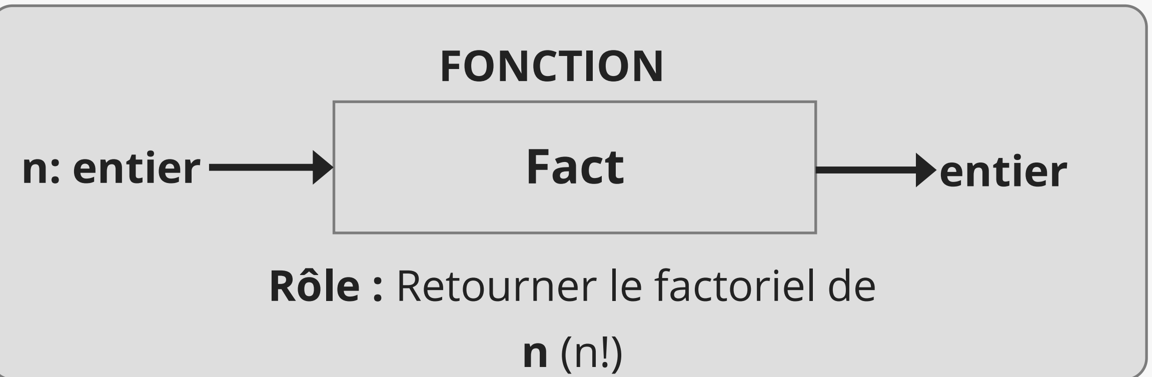
Exemple: Calcul de Combinaisons

Les Fonctions

1ère étape: Découpage modulaire

Fonction : Fact

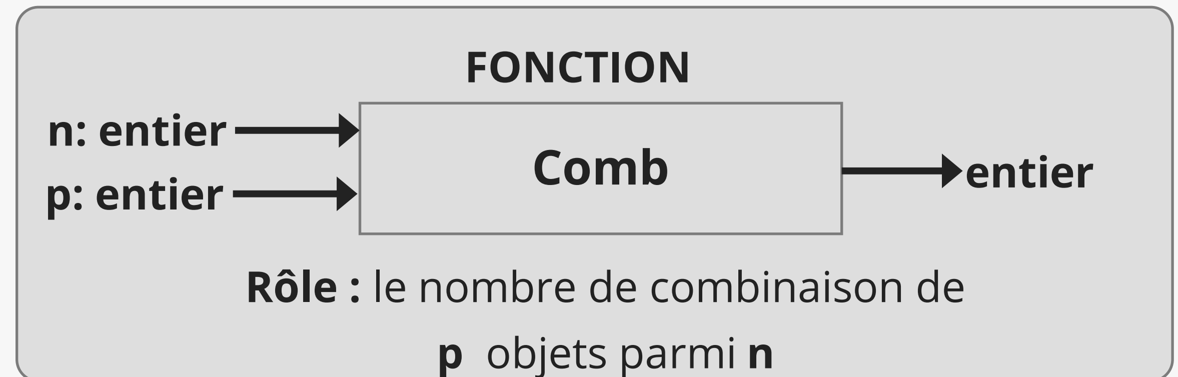
Fact (n) = n!
= n*(n-1)*….*3*2*1



Fonction : Comb

Comb (n,p) $= C_n^p = \frac{n!}{p!*(n-p)!}$

= Fact(n)/Fact(p)*Fact(n-p)



Exemple: Calcul de combinaisons

2ème étape: Construction des modules

```
Fonction Fact (n: entier): entier;
Var        F, i: entier;
Début
   F ← 1;
   Pour i ← 1 à n Faire
          F ← F * i;

    Fact ← F;
Fin;
```

```
Fonction Comb (n , p: entier): entier;
Fonctions :   Fact;
Début
   Comb ← Fact(n)/ Fact(p)* Fact(n-p);
Fin;
```

Les Fonctions

3ème étape: Algorithme principale

```
Algorithme Calcul_comb;
Var        x,y,c: entier;
Fonctions : Comb;
Début
    Lire(x,y);
    c ← Comb(x,y);
    Ecrire ('Le nombre de combinaison = ', c);
Fin;
```

Déclaration d'une Fonction

PASCAL

FUNCTION *nom_fonction* (*Paramètre d'entrés*)**:** *type_retour*;
var {*Déclaration des données locales*}
begin
-
- {*Instructions*}
-

nom_fonction ← *valeur_retour*;
fin;

```
function Fact(n: integer) : integer;
 var F,i : integer;
 begin
   F := 1;
   for i := 2 to n do
       F := F*i;

   Fact := F;
 end;
```

C

type_retour *nom_fonction* (*Paramètre d'entrés*)
{ {*Déclaration des données locales*}
-
- {*Instructions*}
-
-

return *valeur_retour*;
}

```
int Fact (int n)
{
    int F , i;
    F = 1;
    for (i=2; i<=n; i++){
        F = F*i;
    }

    return F;
}
```

Déclaration d'une Fonction

PASCAL

- ✓ En PASCAL, les fonctions et procédures doivent être déclarés **avant** le programme principale.
- ✓ En général, chaque module appelé doit être construit **avant** le module appelant pour qu'il soit **reconnu**.
- ✓ Dans cet exemple:
  - ❑ Un ***Appel*** d'une fonction Fact (ligne **17**)
  - ❑ **n** : ***paramètre*** (paramètre ***formel***) (ligne **5**)
  - ❑ **x** : ***argument*** (paramètre ***effectif***)(ligne **17**)

```
1  program factoriel;
2   uses Crt;
3   var x : integer;
4
5   function Fact(n: integer) : integer;
6    var F,i : integer;
7    begin
8      F := 1;
9      for i := 2 to n do
10         F := F*i;
11
12     Fact := F;
13   end;
14
15 begin
16   Readln(x);
17   Writeln(x,'! =', Fact(x));
18 end.
```

Déclaration d'une Fonction

Programmation C / PASCAL

C

- ✓ En langage C, les fonctions et procédures peuvent être déclarés **avant** et **après** la fonction *main*.
- ✓ Si la fonction est placée **avant** le *main* , le compilateur vérifie les paramètres est exécute la fonction.
- ✓ Si la fonction est placée **après** le *main* , on est besoin de définir un **prototype** de la fonction pour qu'elle soit reconnue.

```
1  #include <stdio.h>
2
3  int Fact (int n)
4  {
5      int F , i;
6      F = 1;
7      for (i=2; i<=n; i++){
8          F = F*i;
9      }
10
11     return F;
12 }
13
14 int main()
15 {
16     int x;
17     scanf("%d", &x);
18     printf("%d! = %d", x, Fact(x));
19
20     return 0;
21 }
```

Déclaration d'une Fonction

C

- ✓ **Un prototype** est une déclaration de fonction pour qu'elle soit utilisée (appelée) même avant sont code.
- ✓ Le prototype est placé au **début du programme** (juste après la déclaration des bibliothèques).
- ✓ Un prototype se déclare comme une fonction

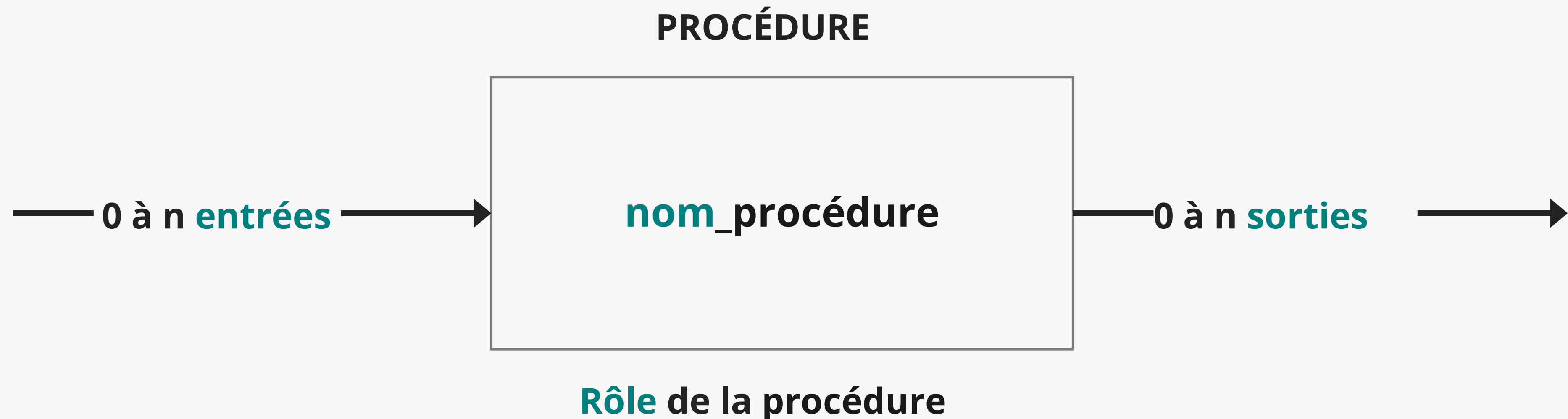
type_retour nom_fonction (Paramètre d'entrés)

```c
#include <stdio.h>

int Fact (int n);

int main()
{
    int x;
    scanf("%d", &x);
    printf("%d! = %d", x, Fact(x));

    return 0;
}

int Fact (int n)
{
    int F , i;
    F = 1;
    for (i=2; i<=n; i++){
        F = F*i;
    }

    return F;
}
```

Les Procédures

Définition & Description

✓ Une procédure est sous-programme (module) qui retourne 0 à n résultat (sortie multiple) de type simple ou composé. Elle peux recevoir 0 à n paramètres d'entrées.



Description d'une procédure

Structure & Syntaxe

Entête

Procédure nom_procédure (Liste des paramètres formels d'entrée et de sortie :Type);

Corps

Type –
Const –
Var –

Déclaration des données (objets) locales

Début
-
-
-
-
-

Traitements

Fin;

Propriétés & Remarques

- ✓ Le corps d'une procédure peut contenir toutes les **déclarations** (Type, Const, Var, ..) et les **structures** algorithmiques (Affectation , Répétition, Conditionnel…).
- ✓ Les **paramètres formels** décrivent les paramètres d'**entrées** et de **sorties** utilisées dans la procédure ainsi que leur **type** et leur **mode de passage**.
- ✓ Dans les procédures, les paramètres formels de **sorties** doivent toujours être décrites en mode de passage **par variable.**

Appels

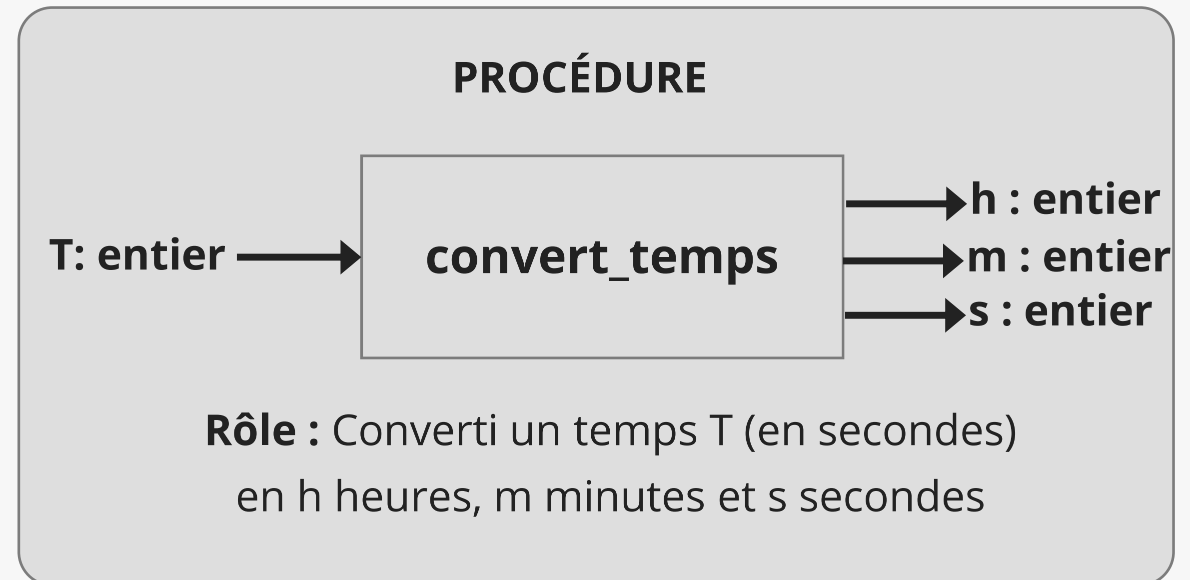
- ✓ L'appel d'une procédure est une **action primitive**. Il est composé du nom de la procédure suivi entre parenthèse de la liste des paramètres effectifs d'entrée et sortis séparés par des virgules.
- ✓ Comme pour les fonctions le **nombre** , l'**ordre** et le **type** des paramètres effectifs doivent être identiques à ceux des paramètres formels.
- ✓ **Exemples:**
  - ❑ remplir_tab (n, T)
  - ❑ echange (x, y)

1ère étape: Découpage modulaire

Procédure: convert_temps

convert_temps (T, h, m, s)

1. On divise T sur 360 : le quotient est h
2. Le reste de cette division est divisé sur 60
   a. Le quotient est m
   b. Le reste est s



2ème étape: Construction des modules

```
Procédure convert_temps (T: entier; VAR h, m, s : entier);
Var      R: entier;
Début
    h ← T DIV 3600;
    R ← T MOD 3600;
    m ← R DIV 60;
    s ← R MOD 60;
Fin;
```

3ème étape: Algorithme principale

```
Algorithme Convert;
Var            A, x, y, z: entier;
Procédures : convert_temps;
Début
    Lire(A);
    convert_temps(A, x, y, z);
    Ecrire (A,'=',x,'heures et ',y,' minutes et ',z,'secondes');
Fin;
```

Exemple: Convertir un temps T (en secondes) en heures, minutes et secondes

PASCAL

PROCEDURE nom_procedure (Paramètre d'entrés/sorties);

var {Déclaration des données locales}

begin

\- {Instructions}

\-

\-

\-

fin;

```
procedure convert_temps(T: integer; VAR h,m,s: integer);
 var R : integer;
 begin
   h := T DIV 3600;
   R := T MOD 3600;
   m := R DIV 60;
   s := R MOD 60;
 end;
```

C

void nom_fonction (Paramètre d'entrés/sorties)

{ {Déclaration des données locales}

\- {Instructions}

\-

\-

\-

\-

}

```
void convert_temps (int T, int *h, int *m, int *s)
{
     int R;

     *h = T / 3600;
     R = T % 3600;
     *m = R / 60;
     *s = R % 60;
}
```

Déclaration d'une Procédure

PASCAL

Programmation C / PASCAL

✓ Dans cet exemple:

- ❑ Un ***Appel*** d'une procédure convert_temps (ligne **5**)
- ❑ **T** : ***paramètre*** (paramètre ***formel*** d'entrée) (ligne **5**)
- ❑ **h,m,s** : ***paramètre*** (paramètre ***formel*** de sortie) (ligne **5**)
- ❑ **A** : ***argument*** (paramètre ***effectif***) d'entrée (ligne **17**)
- ❑ **x,y,z** : ***argument*** (paramètre ***effectif***) de sortie (ligne **17**)

```pascal
1  program Convert;
2   uses Crt;
3   var A,x,y,z : integer;
4
5   procedure convert_temps(T: integer; VAR h,m,s: integer);
6    var R : integer;
7    begin
8      h := T DIV 3600;
9      R := T MOD 3600;
10     m := R DIV 60;
11     s := R MOD 60;
12   end;
13
14 begin
15   Readln(A);
16   convert_temps(A, x, y, z);
17   Writeln(A,'=',x,'heures et ',y,' minutes et ',z,'secondes');
18 end.
```

Déclaration d’une Procédure

C

- ✓ En langage C, le type de retour des procédures est spécifié à *void*.
- ✓ Lors de la déclaration de la procédure, les paramètres formels de sortie sont précédés par *.
- ✓ Lors de l’appel de cette procédure, les paramètres effectifs de sortie sont précédés par **&.**

```c
#include <stdio.h>

void convert_temps (int T, int *h, int *m, int *s);

int main()
{
    int A, x, y, z;
    scanf("%d", &A);
    convert_temps(A, &x, &y, &z);
    printf("%d = %d heures et %d minutes et %d secondes", A, x, y, z);

    return 0;
}

void convert_temps (int T, int *h, int *m, int *s)
{
    int R;

    *h = T / 3600;
    R = T % 3600;
    *m = R / 60;
    *s = R % 60;
}
```