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 2

La Récursivité

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

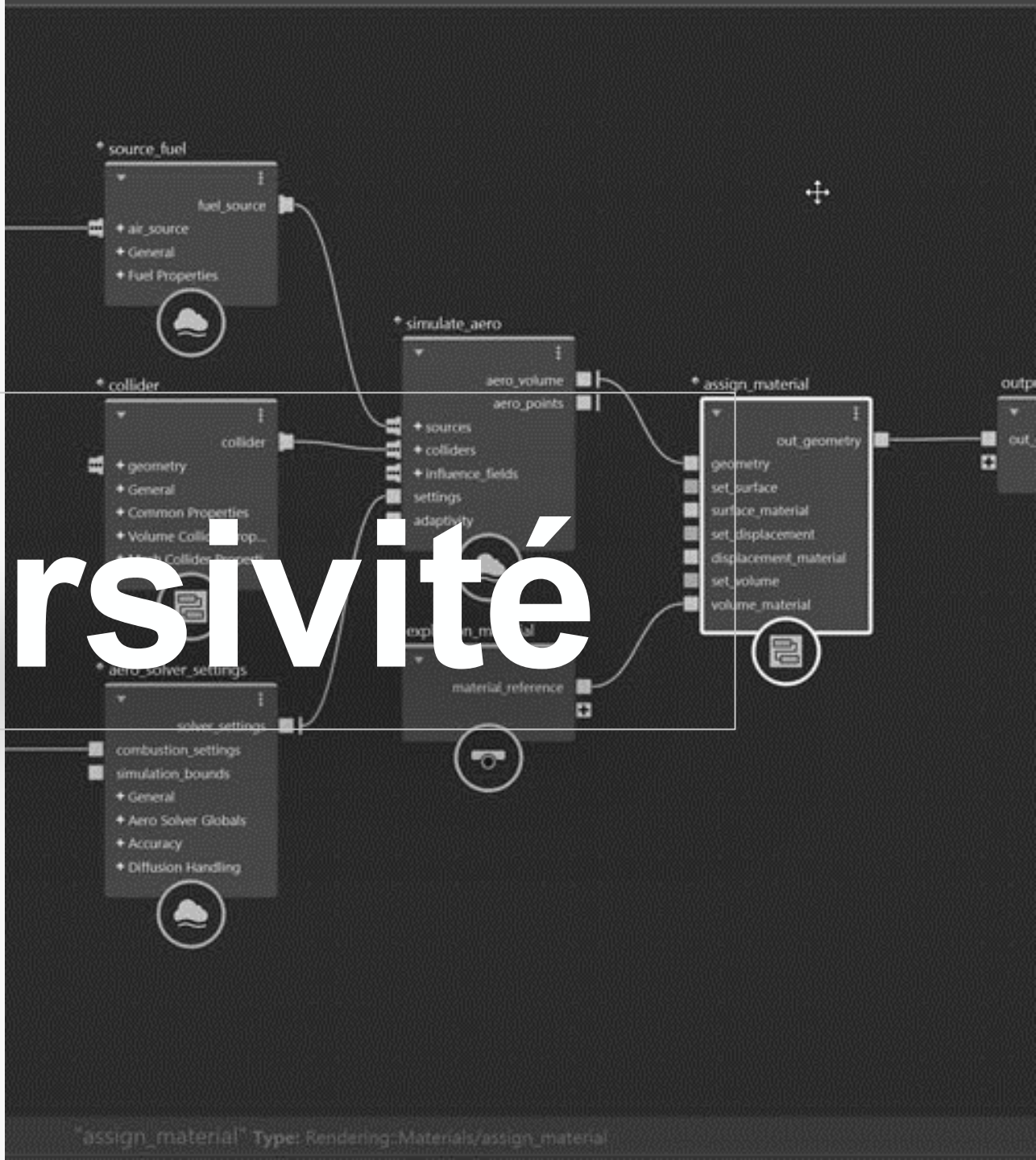
Noureddine AZZOUZA

n.azzouza@univ-dbkm.dz

Plan du Cours

1. Définition de la récursivité
2. Types de récursivité
3. Propriétés de la récursivité
4. Exemples
5. Conception des Algorithmes récursives

La Récursivité



Définition

- ✓ La **récursivité** est le cas lorsque le définition d'une partie d'un tout fait appel au tout.
- ✓ Une procédure ou une fonction est dite **récursive** si elle fait appel à elle-même, ***directement*** ou ***indirectement***.

Appel récursif ------> Boucle

Exemple

Calcul de Factoriel

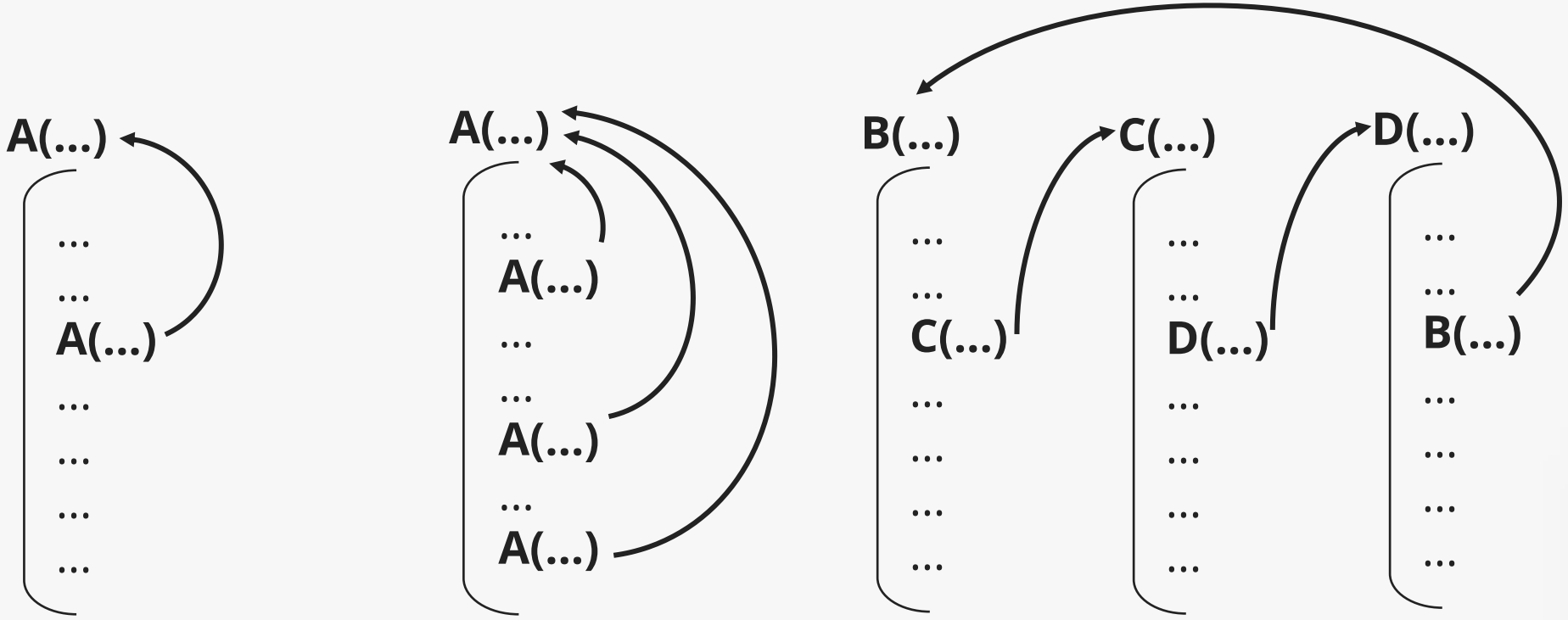
Trouver le factoriel d'un entier n. que $n! = n * (n-1) * (n-2) * \cdots * 2 * 1$

```
Fonction Fact (n : entier): entier;
Début
    Si n = 0 Alors
        Fact ← 1;
    Sinon
        Fact ← n * Fact (n-1);
Fin;
```

Types de Récursivité

- ✓ **Récursivité directe** : c'est lorsqu'un algorithme appelle lui même.
  - ✓ ***Simple*** : s'appelle lui-même une seule fois au maximum
  - ✓ ***Multiple*** : s'appelle elle-même plus d'une fois.
- ✓ **Récursivité indirecte** ou **Croisée**: lorsqu'un algorithme appelle un autre qui appelle le premier. L'algorithme s'appelle indirectement, via une ou plusieurs autres fonctions.

Types de Récursivité



Récursivité Directe Simple

Récursivité Directe Multiple

Récursivité Indirecte (Croisée)

Propriétés de la récursivité

✓ Le corps d'un algorithme récursif doit contenir:

- ❑ Un ou plusieurs **cas particulier** : (**cas d'arrêt**) C'est les cas simple qui nécessite pas d'appels récursifs (entrainent la fin de la récursivité).
- ❑ Un ou plusieurs **cas généraux** : Ce sont les cas complexes qui sont résolus par des appels récursifs (provoquant les appels récursifs).
- ❑ Un ou plusieurs **cas d'erreur**: Ce sont les cas pour lequels la fonction n'est pas définie.

Propriétés de la récursivité

- ✓ Pour éviter les boucles infinies, les appels récursifs doivent être conditionnels (à l'intérieur d'un SI ou un SINON…etc)

- ✓ Les appels récursifs d'un cas général doit toujours mener vers un des cas particulier.

Conception des Algorithmes récursives

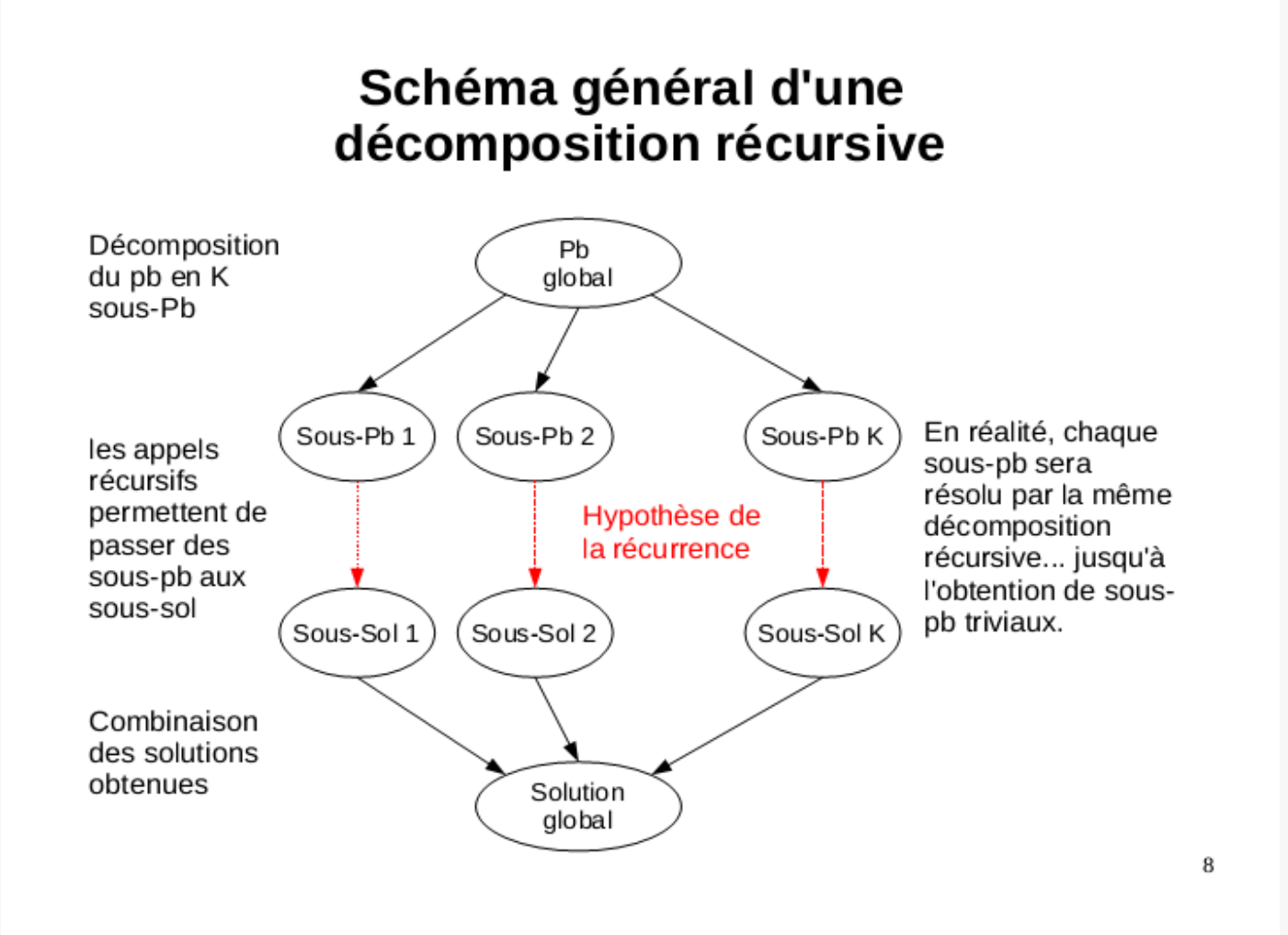
- ✓ La récursivité est un outil puissant
  - ❑ permet de concevoir des solutions (élégantes) sans se préoccuper des détails algorithmiques internes

- ✓ Approche descendante par « décomposition »
  - ❑ spécifier la solution du problème en fonction de la (ou les) solution(s) d'un (ou de plusieurs) sous- problème plus simple(s).

Conception des Algorithmes récursives

- ✓ pour trouver une solution récursive à un problème, on cherche à le décomposer en plusieurs sous problèmes de même type mais de taille inferieur. La méthode générale étant la suivante :
  1. Paramétrage du problème : on détermine les éléments dont dépend la solution et qui caractérisent la taille du problème.
  2. Recherche d'un cas trivial et de sa solution.
  3. Décomposition du cas générale en cas plus simples, eux même décomposables pour aboutir au cas trivial.

Conception des Algorithmes récursives



Les Fonctions

Exemple 1 : La Factorielle

1ère étape: Analyse et Découpage modulaire

Calcul de Factoriel

La factorielle d'un nombre n donné est le produit des nombres entiers inférieurs ou égales à ce nombres

Fonction : Fact

Fact (n) = n!
= n*(n-1)*....*3*2*1

$$\text{Fact (n) = n! = } \begin{cases} 1 & si\ n = 0 \\ n * (n-1)! & si\ n > 0 \end{cases}$$

Rôle : Retourner la factorielle de **n** (n!)

2ème étape: Construction des modules

- ✓ Pour écrire une fonction factorielle récursive, les cas sont les suivants:
  - ❑ ***Cas d'erreur :*** n < 0,
  - ❑ ***Cas d'arrêt :*** n ∈ {0,1}
  - ❑ ***Cas général :*** n! = n * (n-1)!

```
Fonction Fact (n : entier): entier;
Début
   Si n < 0 Alors
      F ← -1;
      écrire (« erreur : n négatif»);
   Sinon
      Si n = 0 ou n = 1 Alors      F ← 1;
      Sinon                        F ← n * Fact (n-1);
 Fact ← F;
Fin;
```

Cas d'erreur

Cas d'arrêt

Cas général

Exemple 1 : La Factorielle

Exemples

3ème étape: Traduction en Programmes

PASCAL

```pascal
function Fact(n: integer) : integer;
 var F : integer;
 begin
   if (n < 0) then
       begin
           F := -1;
           Writeln('erreur : n négatif')
       end
   else
       if (n = 0) or (n = 1) then  F := 1
       else                        F := n*Fact(n-1);
   Fact := F
 end;
```

C

```c
int Fact(int n)
{  int F = -1;
    if(n<0)
        printf("erreur : n négatif");
    else if(n==0 || n==1)
        F = 1;
    else
        F = n*Fact(n-1);

    return F;
}
```

4ème étape: Déroulement et jeu d'essai

✓ Calculer Fact(n) avec **n = 4**

1- (n=4) 4! 4 * ⇒ 3! (**Appel de Fact(3)**)
2- (n=3) 3! 3 * ⇒ 2! (**Appel de Fact(2)**)
3- (n=2) 2! 2 * ⇒ 1! (**Appel de Fact(1)**)
4- (n=1) 1! 1* ⇒ 0! (**Appel de Fact(0)**)
5- (n=0) 0! ⇒ 1 (**cas particulier**)
4- (n=1) 1! 1* ⇒ 1 = 1 (**Après retour de Fact(0)**)
3- (n=2) 2! 2 * ⇒ 1 = 2 (**Après retour de Fact(1)**)
2- (n=3) 3! 3 * ⇒ 2 = 6 (**Après retour de Fact(2)**)
1- (n=4) 4! 4 * ⇒ 6 = 24 (**Après retour de Fact(3)**)

✓ L'appel Fact(4) a généré 4 appels et 4 retours

✓ L'appel initial Fact(4) retourne donc le résultat : 24

✓ Le dernier appel (Fact(0)) est évalué uniquement par le cas particulier.

Exemple 2 : Suite de Fibonacci

1ère étape: Analyse et Découpage modulaire

Suite de Fibonacci

La suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent. Calculer le Nième élément.

Fonction : Fib

$$U_n = \begin{cases} 1 & si\ n = 0 \\ 1 & si\ n = 1 \\ U_{n-1} + U_{n-2} & si\ n > 1 \end{cases}$$

Rôle : Calcule le nième élément dans la suite de Fibonacci

2ème étape: Construction des modules

- ✓ Pour écrire une fonction « Fibonacci » récursive, les cas sont les suivants:
  - ❑ ***Cas d'erreur :*** n < 0,
  - ❑ ***Cas d'arrêt :*** n ∈ {0,1}
  - ❑ ***Cas général :*** $U_n = U_{n-1} + U_{n-2}$

```
Fonction Fib (n : entier): entier;
Début
   Si n < 0 Alors
      F ← -1;
      écrire (« erreur : n négatif»);
   Sinon
      Si n = 0 ou n = 1 Alors  F ← 1;
      Sinon                    F ← Fib(n-1) + Fib(n-2);
 Fib ← F;
Fin;
```

Cas d'erreur

Cas d'arrêt

Cas général

Exemple 2 : Suite de Fibonacci

3ème étape: Traduction en Programmes

PASCAL

```pascal
function Fact(n: integer) : integer;
 var F : integer;
 begin
   if (n < 0) then
       begin
           F := -1;
           Writeln('erreur : n négatif')
       end
   else
       if (n = 0) or (n = 1) then  F := 1
       else                        F := n*Fact(n-1);
   Fact := F
 end;
```

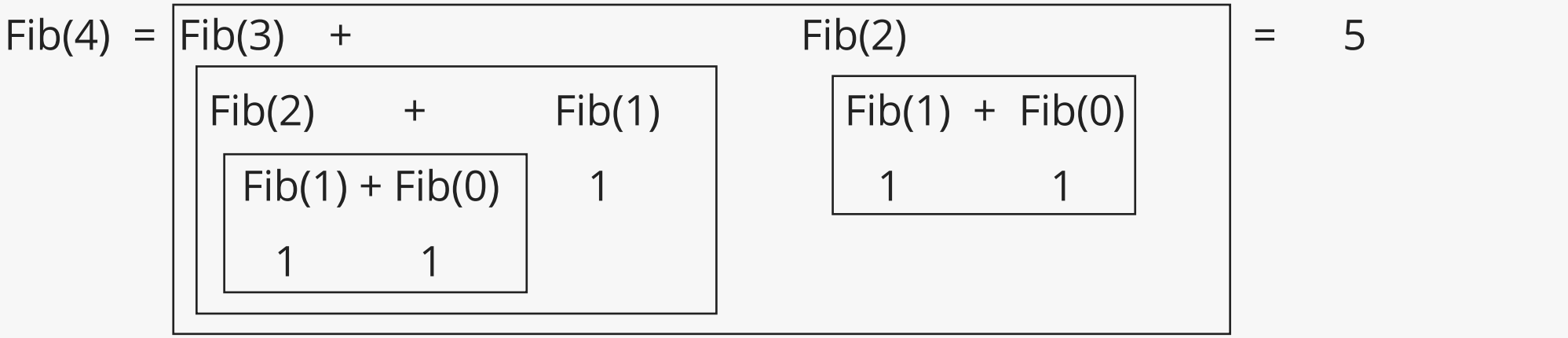
C

```c
int Fact(int n)
{  int F = -1;
    if(n<0)
        printf("erreur : n négatif");
    else if(n==0 || n==1)
        F = 1;
    else
        F = n*Fact(n-1);

    return F;
}
```

4ème étape: Déroulement et jeu d'essai

- ✓ Calculer Fib(n) avec **n = 4**



- ✓ L'appel Fib (4) a généré 8 appels et 8 retours
- ✓ L'appel initial Fib(4) retourne donc le résultat : 5

Exemple 3 : Recherche dichotomique

1ère étape: Analyse et Découpage modulaire

Recherche dichotomique : recherche binaire dans une table ordonnée

un élément x se trouve soit dans la 1ere moitié soit dans la 2e moitié pour le savoir on compare par rapport au milieu.

Fonction : Rech

x = 60

T	10	17	21	38	44	49	60	72	97
	1	2	3	4	5	6	7	8	9

Borne inférieure bi — **milieu** — **Borne supérieure bi**

FONCTION

Rôle : Rechercher la valeur **x** dans le tableau **T** d'entier entre les deux bornes **bi** et **bs**

2ème étape: Construction des modules

Exemples

✓ Pour écrire une « Recherche » dichotomique récursive, les cas sont les suivants:

- ❑ **Cas**
- ❑ **Cas**
- ❑ **Cas**

```
Fonction  Rech (T:Tab; x, bi, bs : entier): entier;
   Result : entier
Début
   Si bi > bs Alors Result ← -1;
   Sinon
      milieu ← (bi + bs) DIV 2;
      Si n = 0 ou n = 1 Alors  F ← 1;
      Sinon                    F ← Fib(n-1) + Fib(n-2);
 Fib ← F;
Fin;
```

Cas d'erreur

Cas d'arrêt

Cas général

2ème étape: Construction des modules

```
Fonction  Rech (T:Tab; x, bi, bs : entier): entier;
    Result, milieu : entier
Début
    Si bi > bs Alors Result ← -1;
    Sinon
        milieu ← (bi + bs) DIV 2;
        Si x = T[milieu]Alors  Result ← -1;
        Sinon
                Si (x < T [milieu] ) Alors
                        Result ← Rech (T, x, bi, milieu-1);
                Sinon
                        Result ← Rech (T, x, milieu+1, bs);
                Fsi
        Fsi
    Fsi
    Rech ← Result
Fin;
```

Cas d'erreur

Cas d'arrêt

Cas général

Cas général