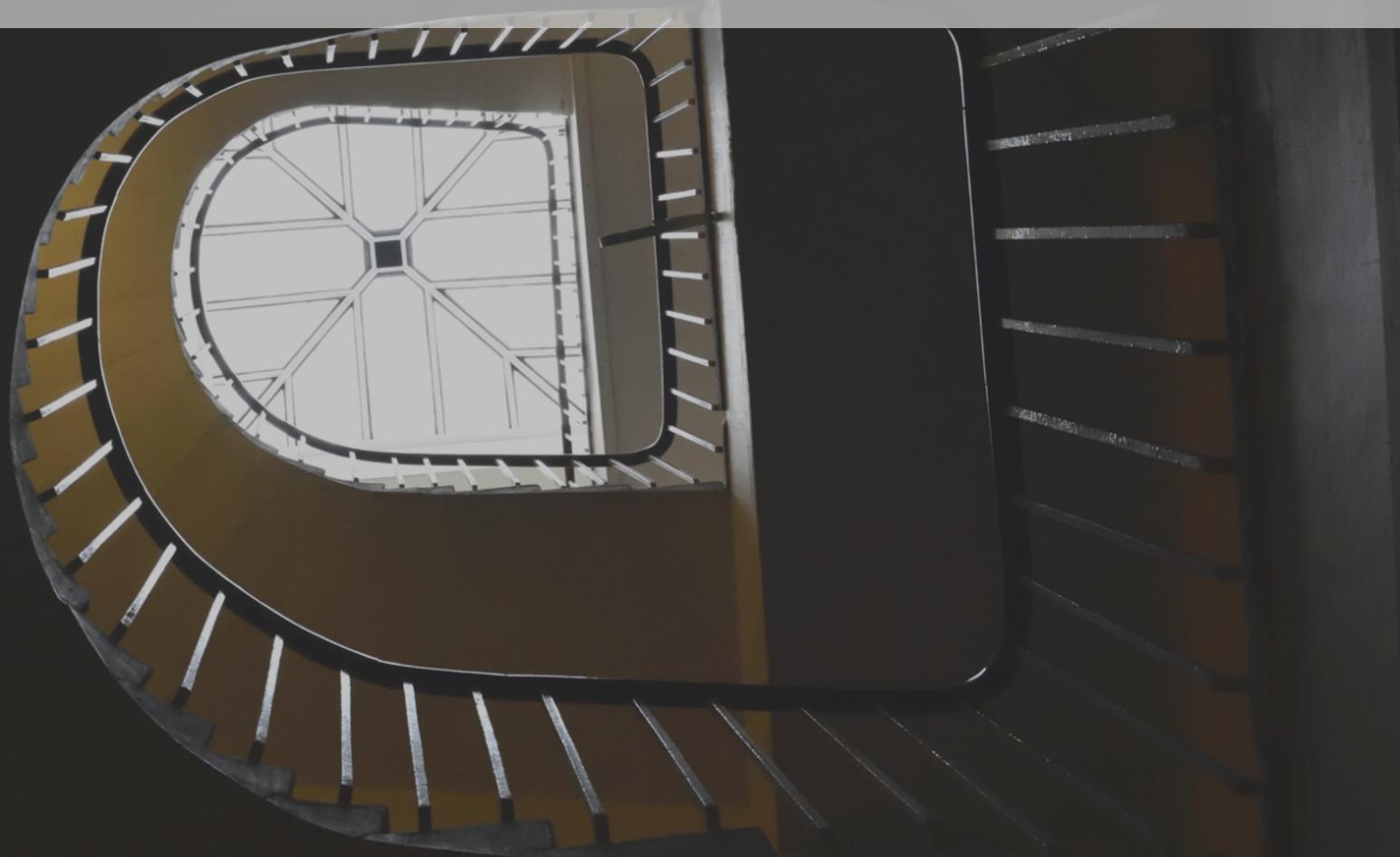


Récursivité



Objectifs

- Rappels pile d'exécution
- Notion de récursivité
- Notion de récursivité terminale

Rappels : pile d'exécution

- La **pile d'exécution** d'un programme est un emplacement **mémoire** qui contient l'ensemble des **paramètres** d'une fonctions, **ses variables locales** et sa **valeur de retour**. Elle contient aussi **l'adresse de retour** de la fonction appelante
 - À chaque appel d'une fonction / méthode sont empilés :
 - un emplacement pour la valeur de retour
 - l'adresse de retour
 - les paramètres
 - Généralement, dans nos représentations schématiques (Pile / Tas), nous ne représentons pas la valeur de retour ni l'adresse de retour
- Comme vous avez pu le constater, la **pile a une capacité limitée**. Si vous la **dépassez**, vous aurez une erreur de type « **StackOverflow** »

Notion de récursivité

- Toute entité est dite récursive si elle se définit à partir d'elle-même
- Plus spécifiquement :
 - Une structure est dit récursive si elle contient un ou des membres de son propre type
 - Exemple : la structure nœud de la liste chainée
 - **Un fonction est dite récursive si elle s'appelle elle-même**
 - Exemple : la fonction partitionner du tri rapide

Notion de récursivité

- Pour définir une fonction récursive, vous avez généralement deux parties :
 - Condition d'arrêt : comment arrêter les appels à soi-même ?
 - Rappel de nous-même avec une variation dans les paramètres
- Factorielle :
$$\begin{cases} f(0) = 1 \\ f(n) = n * f(n - 1) \end{cases}$$
- Des variantes existent :
 - Condition de continuation à la place d'une condition d'arrêt
 - Variations des paramètres dépendants de conditions
 - Etc.

Exemple classique #1

- Factorielle :

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

```
public static int Factorielle_v1(int p_n) {
    if (p_n < 0) {
        throw new ArgumentException("La valeur ne doit pas être négative", nameof(p_n));
    }

    // f(0) = 1
    if (p_n == 0) {
        return 1;
    }

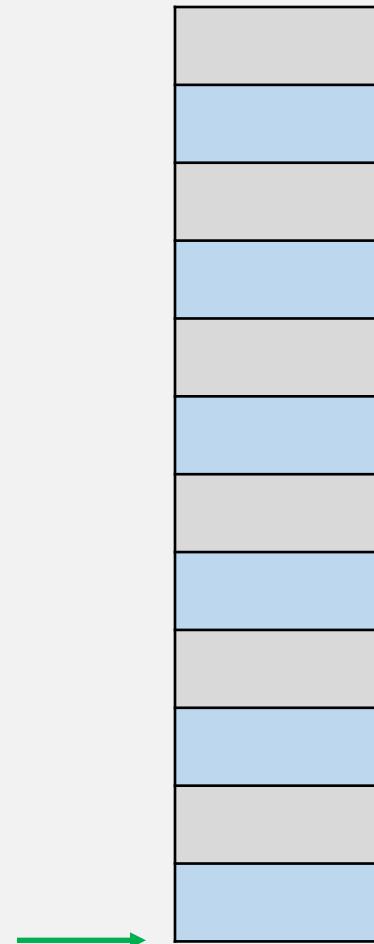
    // f(n) = n * f(n - 1)
    return p_n * Factorielle_v1(p_n - 1);
}
```

Exemple classique #1

- Factorielle :

```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

f(5) :



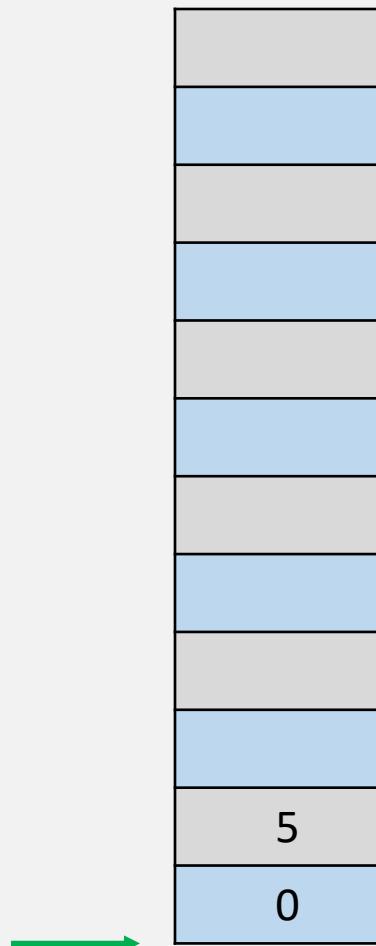
Exemple classique #1

- Factorielle :

```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

$f(5)$:

5 * $f(4)$



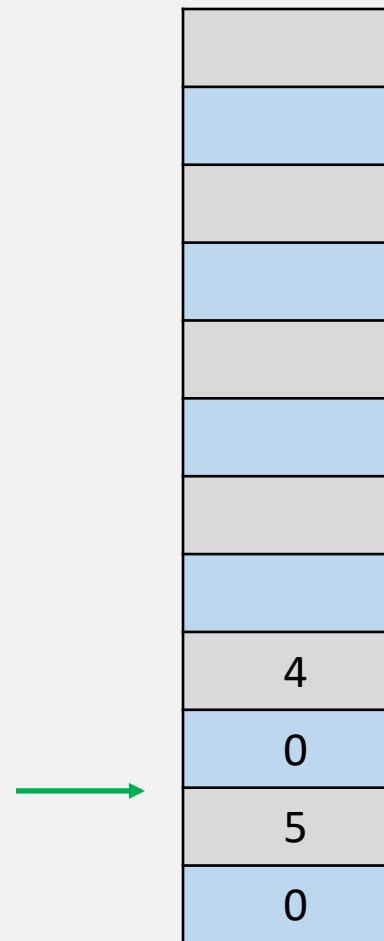
Exemple classique #1

- Factorielle :

```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

$f(5)$:

5 * f(4)
 |
 4 * f(3)



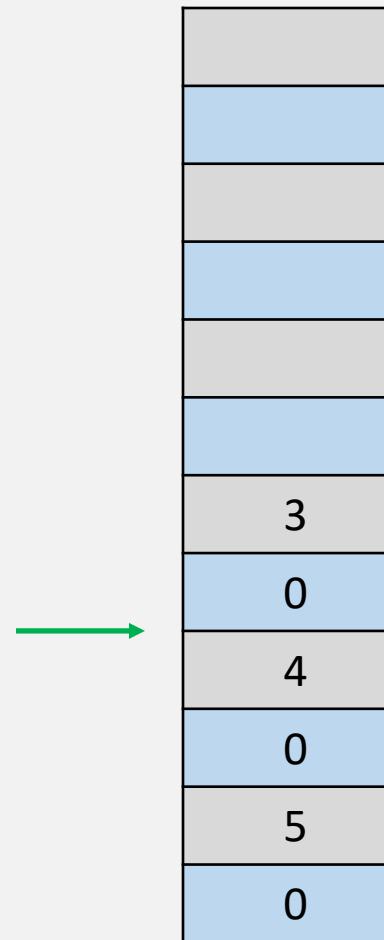
Exemple classique #1

- Factorielle :

```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

$f(5)$:

5 * f(4)
 |
 4 * f(3)
 |
 3 * f(2)



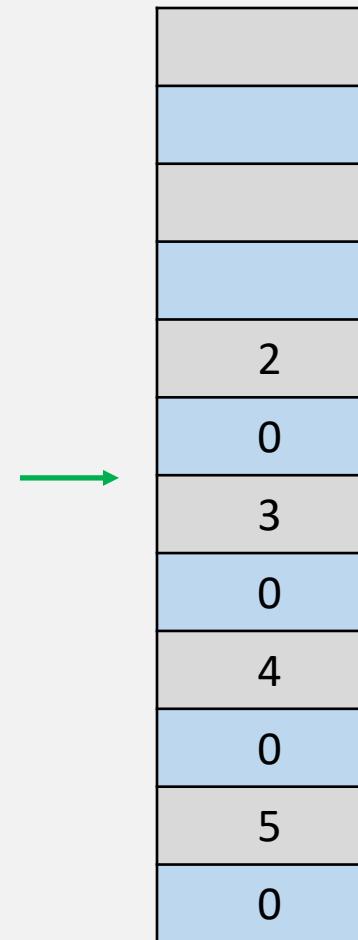
Exemple classique #1

- Factorielle :

```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

$f(5)$:

5 * f(4)
 |
 4 * f(3)
 |
 3 * f(2)
 |
 2 * f(1)



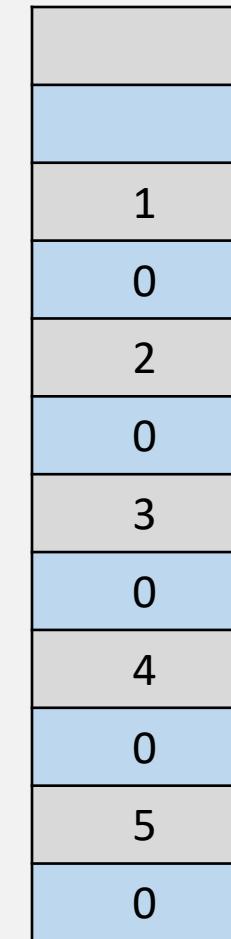
Exemple classique #1

- Factorielle :

```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

$f(5)$:

5 * f(4)
 |
 4 * f(3)
 |
 3 * f(2)
 |
 2 * f(1)
 |
 1 * f(0)



Exemple classique #1

- Factorielle :

```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

f(5) :

```
5 * f(4)  
  4 * f(3)  
    3 * f(2)  
      2 * f(1)  
        1 * f(0)
```



0
0
1
0
2
0
3
0
4
0
5
0

Exemple classique #1

- Factorielle :

```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

f(5) :

```
5 * f(4)  
  4 * f(3)  
    3 * f(2)  
      2 * f(1)  
        1 * f(0)  
          1
```



0
1
1
0
2
0
3
0
4
0
5
0

Exemple classique #1

- Factorielle :

```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

f(5) :

```
5 * f(4)  
  4 * f(3)  
    3 * f(2)  
      2 * f(1)  
        1 * f(0)  
          1
```



0
1
1
0
2
0
3
0
4
0
5
0

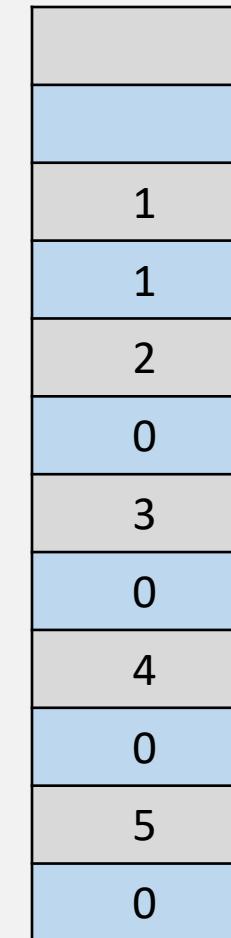
Exemple classique #1

- Factorielle :

```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

$f(5)$:

```
5 * f(4)  
  4 * f(3)  
    3 * f(2)  
      2 * f(1)  
        1 * f(0)  
          1  
        1 * 1 = 1
```



Exemple classique #1

- Factorielle :

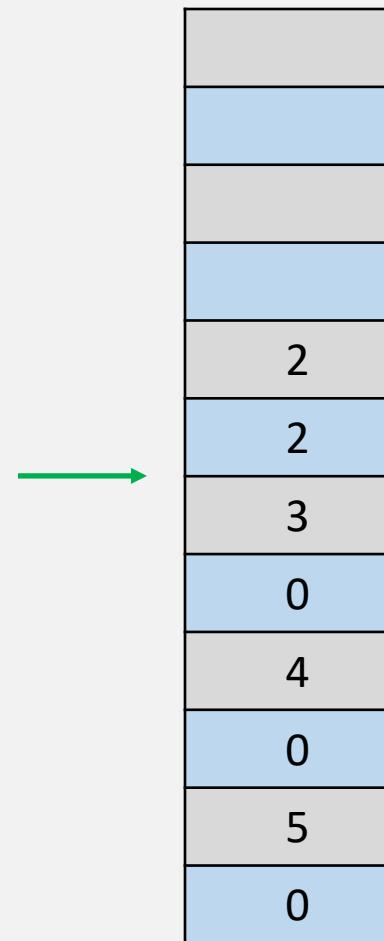
```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

f(5) :

The diagram illustrates the recursive expansion of the expression $5 * f(4)$. It shows a vertical stack of terms, each consisting of a multiplier and a function call, separated by a dotted line:

- $5 * f(4)$
- $4 * f(3)$
- $3 * f(2)$
- $2 * f(1)$
- $1 * f(0)$
- 1
- $1 * 1 = 1$
- $2 * 1 = 2$

The terms are arranged from top to bottom, with the final result at the bottom.



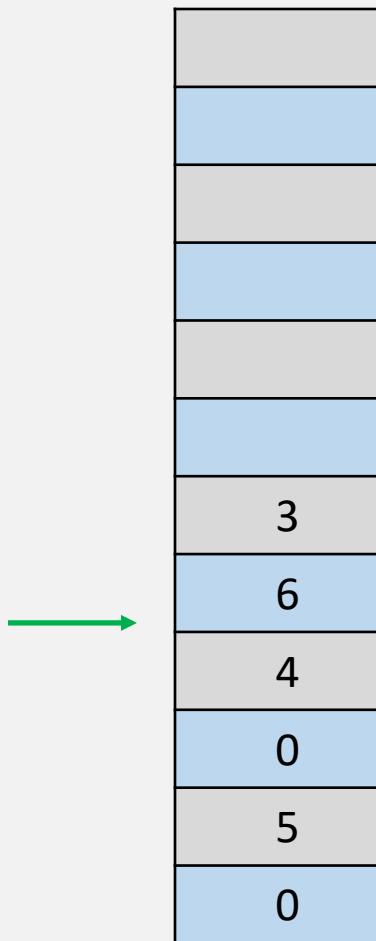
Exemple classique #1

- Factorielle :

```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

$f(5)$:

$$\begin{aligned} & 5 * f(4) \\ & \quad 4 * f(3) \\ & \quad \quad 3 * f(2) \\ & \quad \quad \quad 2 * f(1) \\ & \quad \quad \quad \quad 1 * f(0) \\ & \quad \quad \quad \quad \quad 1 \\ & \quad \quad \quad \quad \quad 1 * 1 = 1 \\ & \quad \quad \quad \quad 2 * 1 = 2 \\ & \quad \quad 3 * 2 = 6 \end{aligned}$$



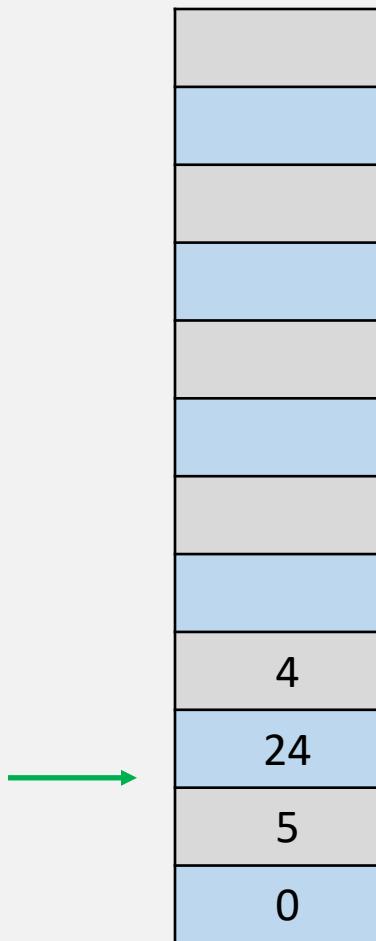
Exemple classique #1

- Factorielle :

```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

$f(5)$:

$$\begin{aligned} & 5 * f(4) \\ & \quad 4 * f(3) \\ & \quad \quad 3 * f(2) \\ & \quad \quad \quad 2 * f(1) \\ & \quad \quad \quad \quad 1 * f(0) \\ & \quad \quad \quad \quad \quad 1 \\ & \quad \quad \quad \quad \quad 1 * 1 = 1 \\ & \quad \quad \quad \quad 2 * 1 = 2 \\ & \quad \quad \quad 3 * 2 = 6 \\ & \quad 4 * 6 = 24 \end{aligned}$$



Exemple classique #1

- Factorielle :

```
if (p_n == 0) {  
    return 1;  
}  
  
return p_n * Factorielle_v1(p_n - 1);
```

$f(5)$:

$$\begin{aligned} & 5 * f(4) \\ & \quad 4 * f(3) \\ & \quad \quad 3 * f(2) \\ & \quad \quad \quad 2 * f(1) \\ & \quad \quad \quad \quad 1 * f(0) \\ & \quad \quad \quad \quad \quad 1 \\ & \quad \quad \quad \quad \quad 1 * 1 = 1 \\ & \quad \quad \quad \quad 2 * 1 = 2 \\ & \quad \quad \quad 3 * 2 = 6 \\ & \quad \quad 4 * 6 = 24 \\ & 5 * 24 = 120 \end{aligned}$$



Exemple classique #1 – V2

- Factorielle :

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

```
public static int Factorielle_v2(int p_n) {
    if (p_n < 0) {
        throw new ArgumentException("La valeur ne doit pas être négative", nameof(p_n));
    }

    return Factorielle_v2_rec(p_n);
}

private static int Factorielle_v2_rec(int p_n) {
    if (p_n == 0) {
        return 1;
    }

    return p_n * Factorielle_v2_rec(p_n - 1);
}
```

Exemple classique #2

- Remplacer une boucle – calculer la somme des nombres de n à 1

```
public static int CalculerSomme1An(int p_n) {
    if (p_n < 1) {
        throw new ArgumentException("La valeur doit être supérieure à 0", nameof(p_n));
    }

    return CalculerSomme1An_rec(p_n);
}

private static int CalculerSomme1An_rec(int p_n) {
    if (p_n == 1) {
        return 1;
    }

    return p_n + CalculerSomme1An_rec(p_n - 1);
}
```

Exemple classique #3

- Remplacer une boucle – afficher les nombres de 1 à n

```
public static void AfficherNombresDe1A_v1(int p_n) {
    if (p_n < 1) {
        throw new ArgumentException("La valeur doit être supérieure à 0", nameof(p_n));
    }

    AfficherNombresDeA_v1_rec(1, p_n);
}

private static void AfficherNombresDeA_v1_rec(int p_de, int p_a) {
    Console.Out.WriteLine(p_de);

    if (p_de < p_a) {
        AfficherNombresDeA_v1_rec(p_de + 1, p_a);
    }
}
```

Exemple classique #3

- Remplacer une boucle – afficher les nombres de 1 à n

```
public static void AfficherNombresDe1A_v2(int p_n) {
    if (p_n < 1) {
        throw new ArgumentException("La valeur doit être supérieure à 0", nameof(p_n));
    }

    AfficherNombresDeA_v2_rec(p_n);
}

private static void AfficherNombresDeA_v2_rec(int p_n) {
    if (p_n > 1) {
        AfficherNombresDeA_v2_rec(p_n - 1);
    }

    Console.Out.WriteLine(p_n);
}
```

Récursivité terminale

- Une fonction est dite **récursive terminale** si la valeur de retour **ne dépend que de l'appel subséquent**, c'est-à-dire si le retour est de la forme « `return f(...)` » ou « `f(...)` » où `f` est le nom de la fonction récursive.

```
private static void AfficherNombres_rec(int p_de, int p_a) {  
    Console.Out.WriteLine(p_de);  
  
    if (p_de < p_a) {  
        AfficherNombres_rec(p_de + 1, p_a);  
    }  
}
```

Récursivité terminale

```
private static int Factorielle_v2_rec(int p_n) {  
    if (p_n == 0) {  
        return 1;  
    }  
  
    return p_n * Factorielle_v2_rec(p_n - 1);  
}
```

Non récursivité terminale

Récurseurité terminale – transformation

```
private static int Factorielle_v2_rec(int p_n) {  
    if (p_n == 0) {  
        return 1;  
    }  
  
    return p_n * Factorielle_v2_rec(p_n - 1);  
}
```

```
public static int Factorielle_v3(int p_n) {  
    if (p_n < 0) {  
        throw new ArgumentException("La valeur ne doit pas être négative", nameof(p_n));  
    }  
  
    return Factorielle_v3_rec(p_n, 1);  
}  
  
private static int Factorielle_v3_rec(int p_n, int p_res) {  
    if (p_n == 0) {  
        return p_res;  
    }  
    return Factorielle_v3_rec(p_n - 1, p_res * p_n);  
}
```