Récursivité[1] : Algorithmes et Complexité
Partie 1

Salem BENFERHAT

Centre de Recherche en Informatique de Lens (CRIL-CNRS)
email : benferhat@cril.fr

[1] Version préliminaire du cours. Tout retour sur la forme comme sur le fond est le bienvenu.

Motivations

- La récursivité est essentielle dans de nombreux langages de programmation

Motivations

- La récursivité est essentielle dans de nombreux langages de programmation
- Elle permet d'offrir des solutions élégantes à de nombreux problèmes

Motivations

- La récursivité est essentielle dans de nombreux langages de programmation
- Elle permet d'offrir des solutions élégantes à de nombreux problèmes
- Elle s'applique particulièrement pour les problèmes qui se décompose en sous-problèmes similaires mais de tailles plus petites. Elle reprend le principe de "diviser pour régner".

Motivations

Définition

Une fonction est dite récursive si elle utilise une référence à elle-même dans sa définition.

3 principes ...

3 principes ...

- Une fonction récursive doit évidement avoir un appel récursif.

3 principes ...

3 principes ...

1. Une fonction récursive doit évidement avoir un appel récursif.
2. Une fonction doit contenir des cas de ”base” qui ne contiennent pas d’appels récursifs. Ca représente l’équivalent de conditions d’arrêt dans des procédure itératives. Sans ces situations de base, la fonction ”boucle”...

3 principes ...

3 principes ...

1. Une fonction récursive doit évidement avoir un appel récursif.
2. Une fonction doit contenir des cas de ”base” qui ne contiennent pas d’appels récursifs. Ca représente l’équivalent de conditions d’arrêt dans des procédure itératives. Sans ces situations de base, la fonction ”boucle”...
3. Les fonctions récursives doivent converger vers ces cas de base pour toutes les valeurs données en entrée.

Récursivité

Hypothèses de travail

- Etablir les cas de base (que l'on sait traiter facilement).

Récursivité

Hypothèses de travail

- Etablir les cas de base (que l'on sait traiter facilement).
  - Ces cas de bases représentent les conditions d'arrêt où on ne fait plus d'appels récursif

Récursivité

Hypothèses de travail

- Etablir les cas de base (que l'on sait traiter facilement).
  - Ces cas de bases représentent les conditions d'arrêt où on ne fait plus d'appels récursif
- Pour écrire une fonction récursive avec un paramètre donné (exemple un entier n), faites l'hypothèse que la fonction est disponible pour les paramètres plus petits (n-1, n-2, etc.)

Types de récursivité

Type de récursivité

Récursivité simple

Un algorithme est dit simplement récursif s'il ne fait qu'un seul appel à lui-même.

Exemple de récursivité simple

La factorielle de n

$$n! = n * (n-1)!,$$

et

$$0! = 1.$$

```
2   int factorielle(int n)
3   {
4     if (n==0) return 1;
5     else return factorielle(n-1)*n;
6   }
```

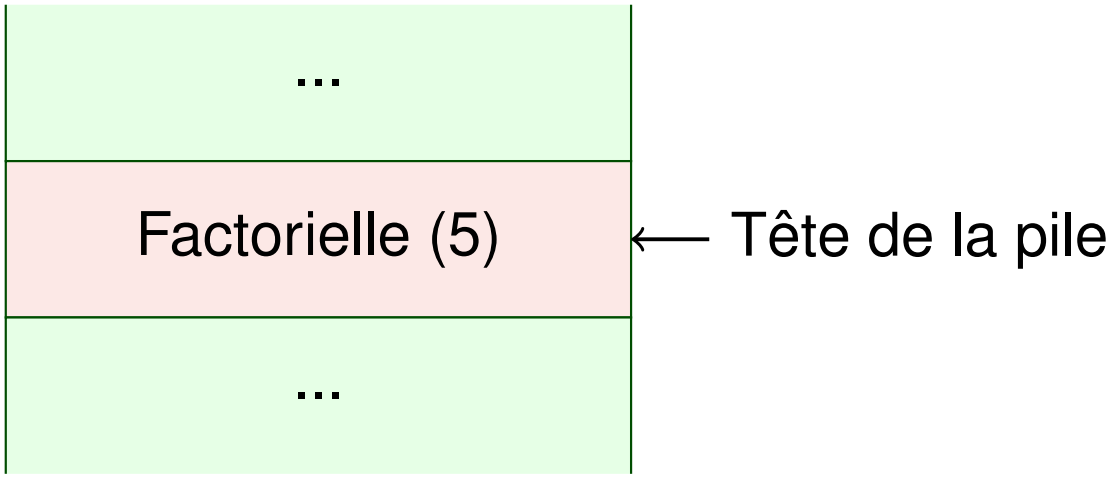
Exécution de la fonction factiorelle

Pile d'execution

La Pile d'exécution (stack) du programme en cours est un emplacement mémoire destiner à mémoriser les paramètres, les variables locales ainsi que les adresses de retour des fonctions en cours d'exécution.

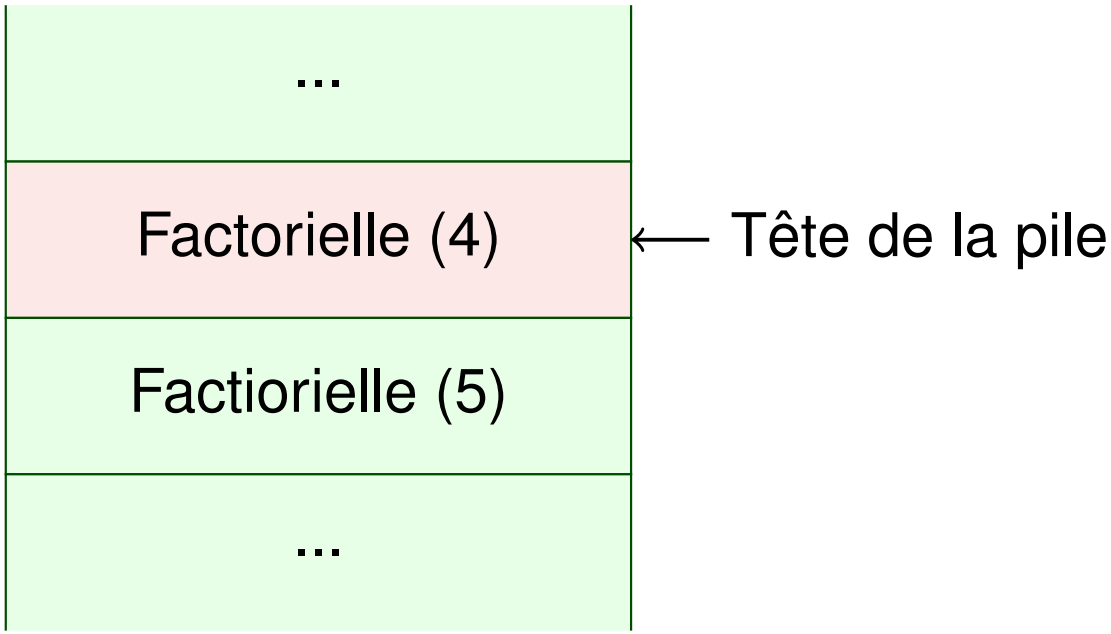
Appel à la fonction récursive : Etape = 1

On empile la fonction factorielle de 5



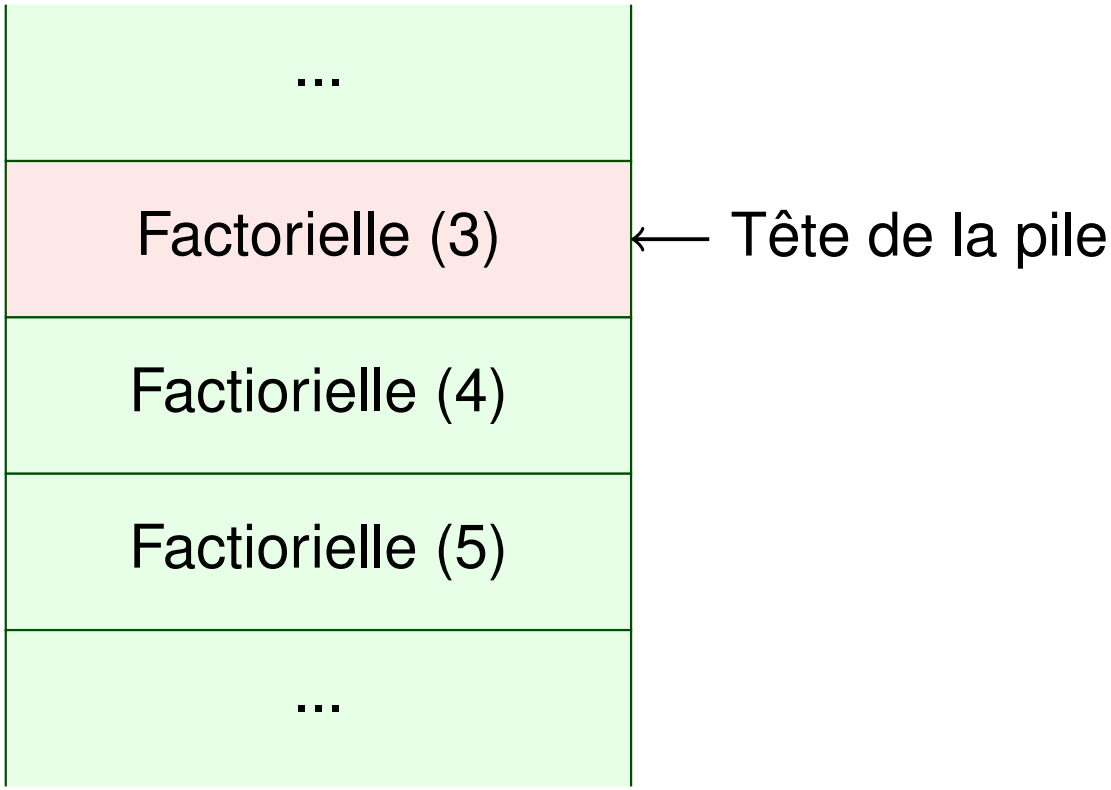
Appel à la fonction récursive : Etape = 2

On empile la fonction factorielle de 4



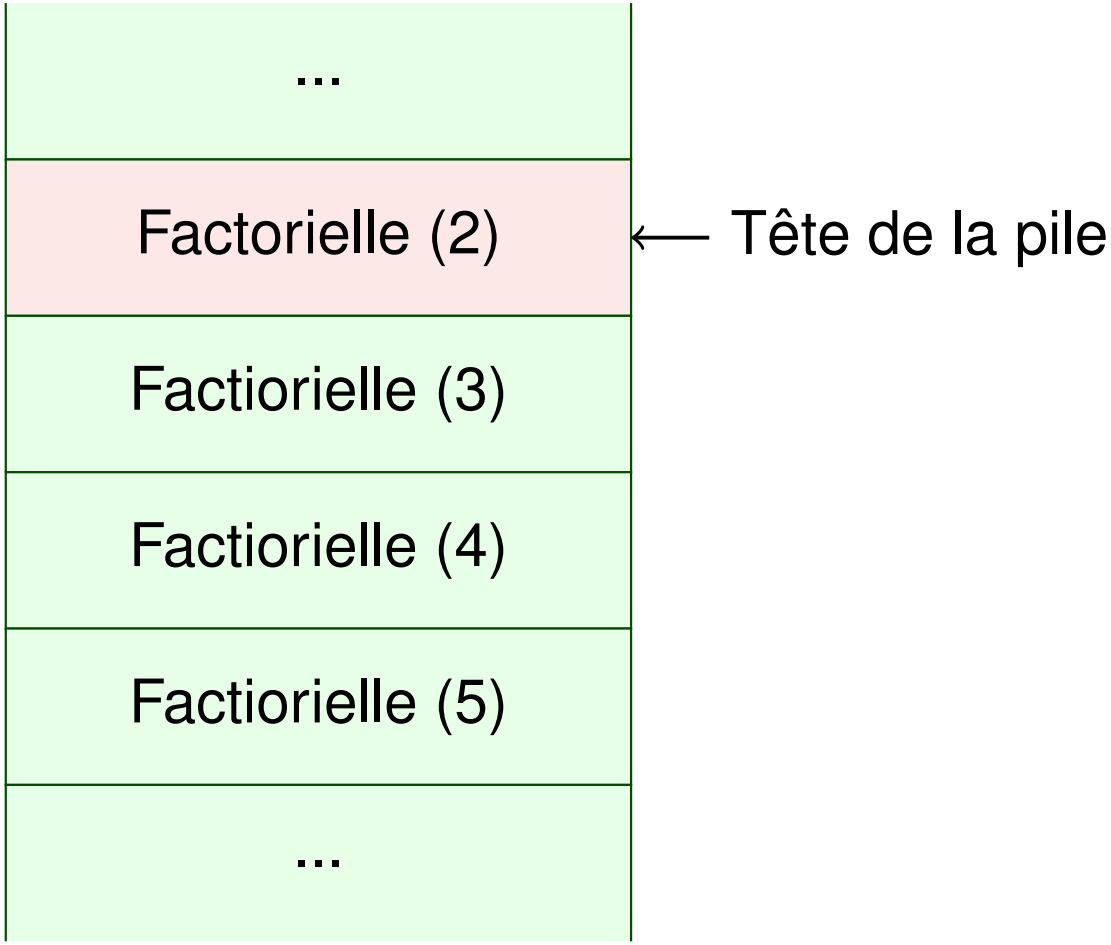
Appel à la fonction récursive : Etape = 3

On empile la fonction factorielle de 3



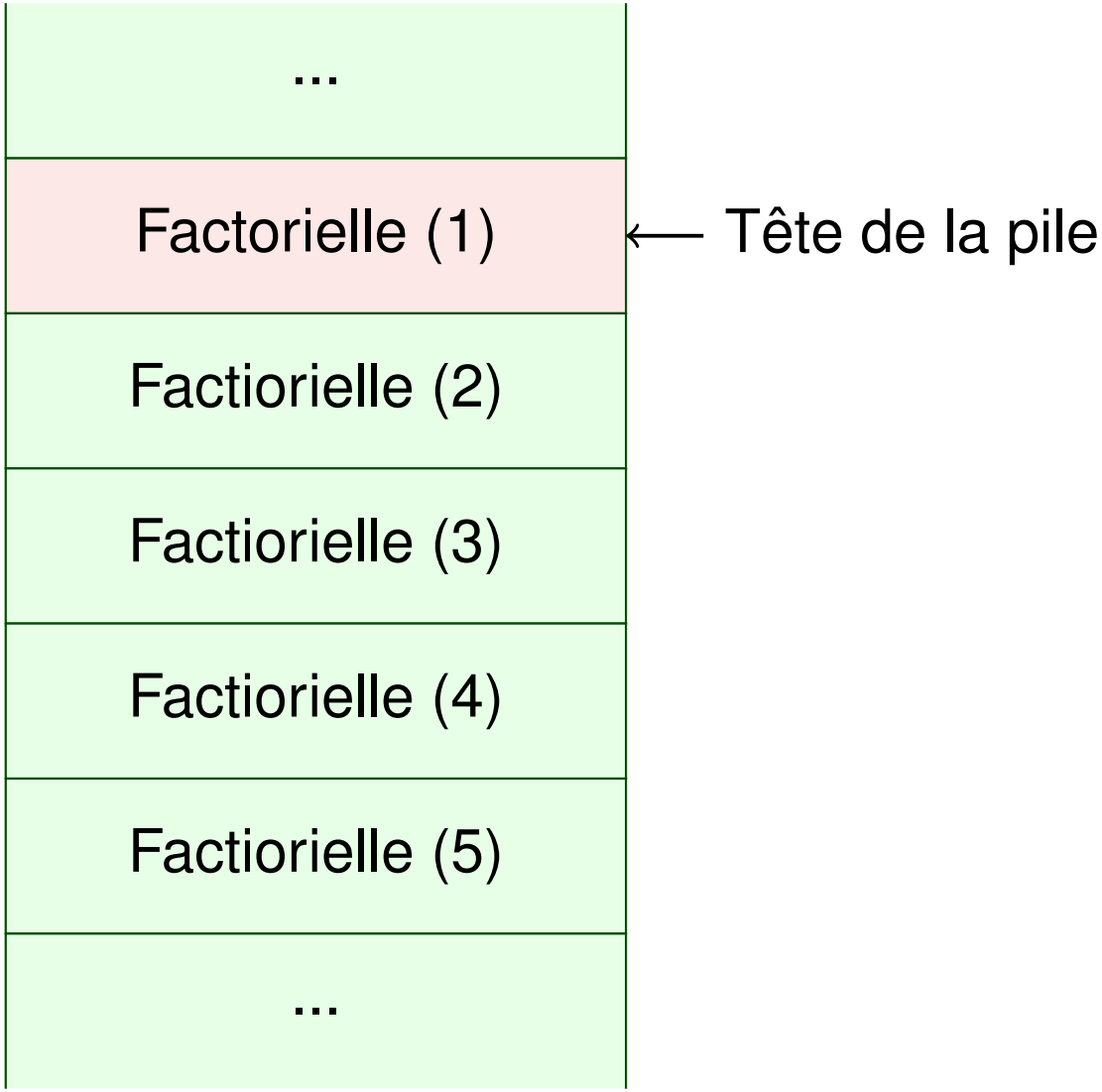
Appel à la fonction récursive : Etape = 4

On empile la fonction factorielle de 2



Appel à la fonction récursive : Etape = 5

On empile la fonction factorielle de 1



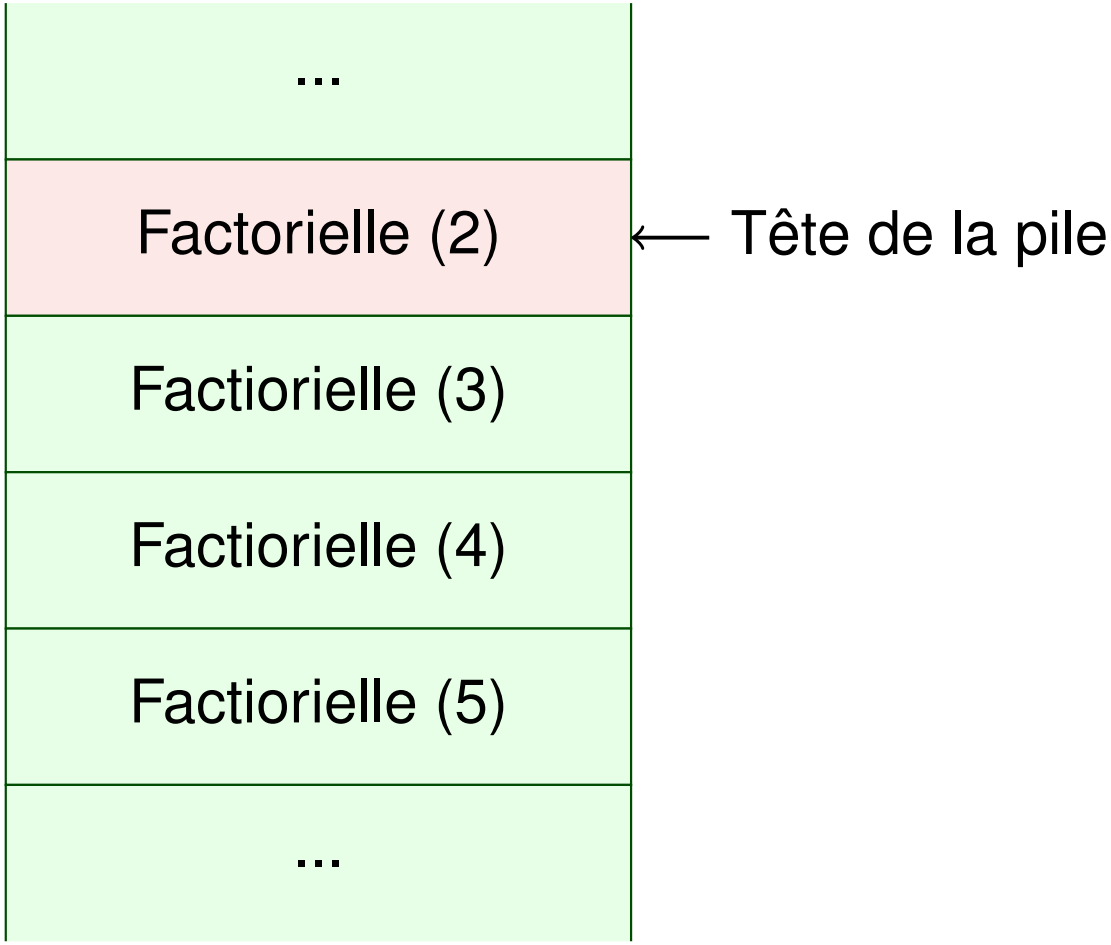
Résultat de la fonction récursive : Factiorelle(1)

On calcule la fonction factorielle de 1

$$\text{Factiorelle}(1) = 1 * \text{Factiorelle}(0) = 1.$$

Retour de la fonction récursive

On depile la fonction factorielle (1)



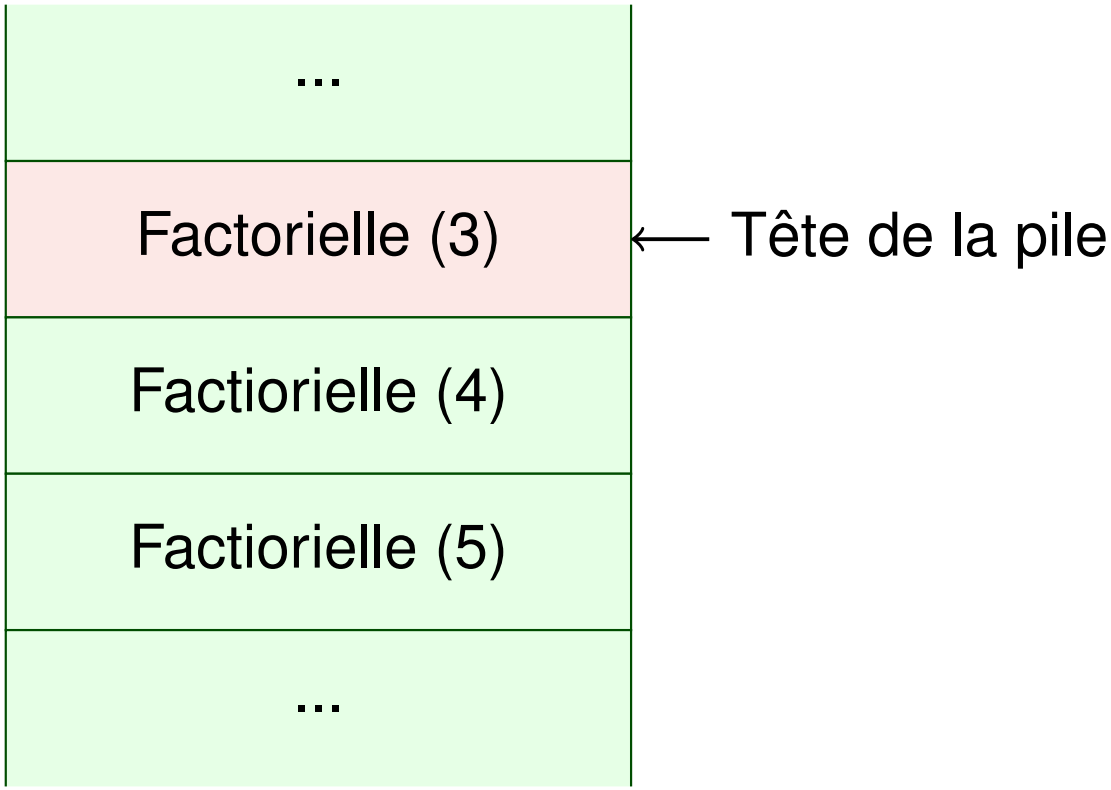
Résultat de la fonction récursive : Factiorelle(2)

On calcule la fonction factorielle de 2

$$\text{Factiorelle}(2) = 2 * \text{Factiorelle}(1) = 2.$$

Retour de la fonction récursive

On depile la fonction factorielle (2)



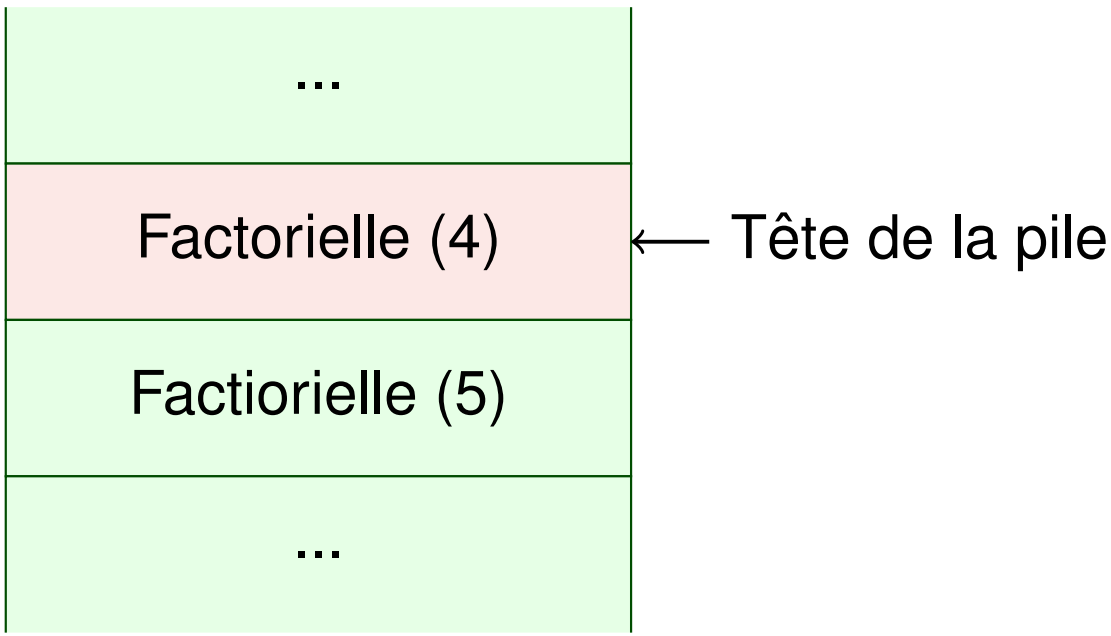
Résultat de la fonction récursive : Factiorelle(3)

On calcule la fonction factorielle de 3

$$\text{Factiorelle}(3) = 3 * \text{Factiorelle}(2) = 6.$$

Retour de la fonction récursive

On depile la fonction factorielle (3)



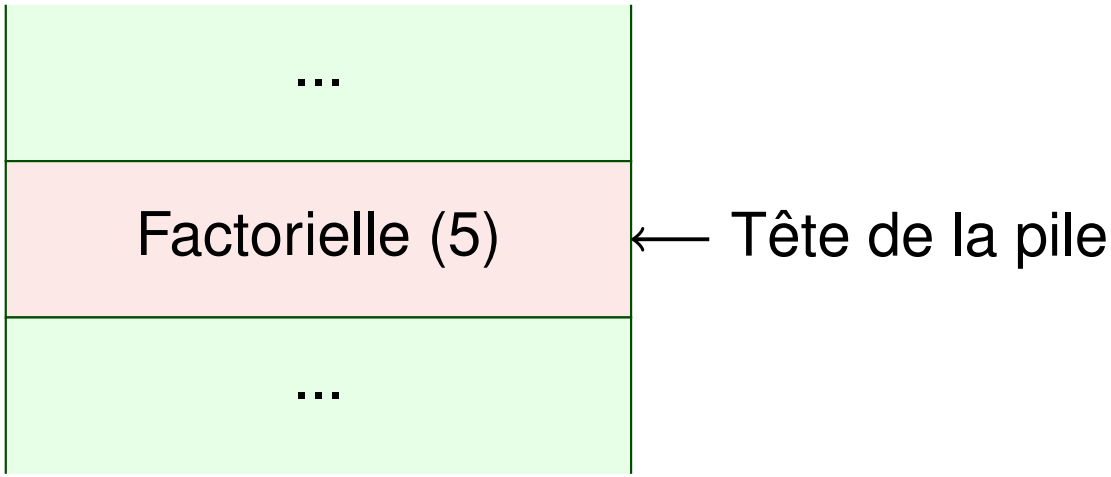
Résultat de la fonction récursive : Factiorelle(4)

On calcule la fonction factorielle de 4

$$\text{Factiorelle}(4) = 4 * \text{Factiorelle}(3) = 24.$$

Retour de la fonction récursive

On depile la fonction factorielle (4)



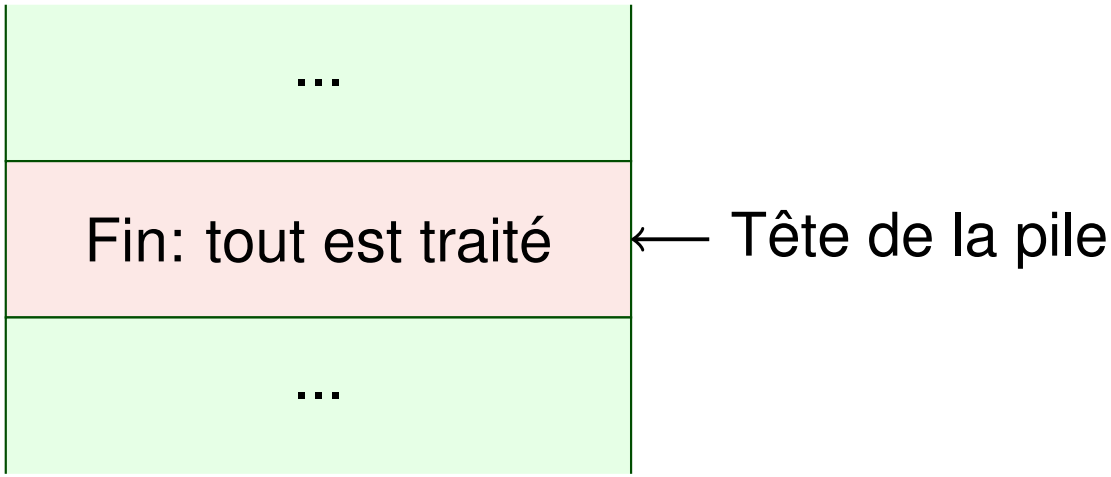
Résultat de la fonction récursive : Factiorelle(5)

On calcule la fonction factorielle de 5

$$\text{Factiorelle}(5) = 5 * \text{Factiorelle}(4) = 120.$$

Retour de la fonction récursive

On depile la fonction factorielle (5)



Complexité de la fonction factorielle

Solution expérimentale

- Une première méthode immédiate est de le programmer, puis de le tester sur un certain nombre de valeurs (les plus représentatifs)?
- On compte par exemple le nombre d'appels récursifs.
- Puis on relie ce nombre d'appels par rapport à la taille des données.

Complexité de la fonction factorielle

Calcul du nombre d'appels récusifs

Utiliser une variable, initialisée à zéro, puis l'incrémenter à l'intérieur de la fonction récursive (par exemple la mettre comme première instruction de la fonction)

```
2   int factorielle(int n)
3   {
4     Nbre_appels_recursif++;
5     if (n==0) return 1;
6     else return factorielle(n-1)*n;
7   }
```

Exécution du programme : nombre d'appels récursif

n	Factorielle(n)	Nombre d'appels récursifs(n)
0	1	1
1	1	2
2	2	3
3	6	4
4	24	5
5	120	6
6	720	7
7	5040	8
8	40320	9
9	362880	10
10	3628800	11
11	39916800	12
12	479001600	13
13	1932053504	14
14	1278945280	15
15	2004310016	16

Tour de Hanoï

Deuxième partie