Algorithmique et Structures de données

Thomas Lavergne & Florent Hivert

Mél : thomas.lavergne@lisn.upsaclay.fr

1 La récursivité

2 Applications

3 En pratique

La récursivité

Une Matriochka est une poupée qui contient soit rien soit une autre matriochka plus petite.

La récursivité

Principe général, si :

- je sais construire l'objet le plus simple,
- à partir d'un objet simple je sais construire un objet un peu plus complexe

alors je sais construire tous les objets.

Récursivité et récurrence

Deux notions très proches :

- mathématiques : récurrence
- informatique : récursivité

De nombreuses définitions mathématiques sont récursives :

Définition (Peano)

- *0 est un entier naturel.*
- *Tout entier n a un successeur unique Sn* $(= n + 1)$ *;*
- *Tout entier sauf* 0 *est le successeur d'un unique entier ;*
- *Pour tout énoncé* $P(n)$ *si* $P(0)$ *est vrai et si pour tout n,* $P(n)$ *implique* $P(Sn)$ *alors l'énoncé* $\forall n : P(n)$ *est vrai.*

Récursivité : définition

Moyen simple et élégant de résoudre certains problèmes.

Définition

*On appelle **récursive** toute fonction ou procédure qui **s'appelle elle-même***

```
unsigned int fact(unsigned int N) {
   if (N == 0) return 1;
   else        return N*fact(N-1);
}
```

Ça marche ! ! !

Récursivité : définition

Moyen simple et élégant de résoudre certains problèmes.

Définition

*On appelle **récursive** toute fonction ou procédure qui **s'appelle elle-même***

```
unsigned int fact(unsigned int N) {
   if (N == 0) return 1;
   else        return N*fact(N-1);
}
```

Ça marche ! ! !

Comment ça marche ?

```
Appel à fact(4)
.   4*fact(3) = ?
.   Appel à fact(3)
.   .   3*fact(2) = ?
.   .   Appel à fact(2)
.   .   .   2*fact(1) = ?
.   .   .   Appel à fact(1)
.   .   .   .   1*fact(0) = ?
.   .   .   .   Appel à fact(0)
.   .   .   .   Retour de la valeur 1
.   .   .   .   1*1
.   .   .   Retour de la valeur 1
.   .   .   2*1
.   .   Retour de la valeur 2
.   .   3*2
.   Retour de la valeur 6
.   4*6
Retour de la valeur 24
```

Point terminal

Retenir

Comme dans le cas d'une boucle, il faut un cas d'arrêt où l'on ne fait pas d'appel récursif.

```
récursive(paramètres) {
    if (TEST_D'ARRET) {
        instructions du point d'arrêt
    } else {
        instructions
        récursive(paramètres changés);  // appel récursif
        instructions
    }
}
```

Encore plus fort

Une fonction peu s'apeller plusieurs fois :

```
unsigned int fibo(unsigned int n) {
    if (n <= 2) return 1;
    else return fibo(n - 1) + fibo(n - 2);
}
```

Attention à l'explosion combinatoires ! ! !

Encore plus fort

Une fonction peu s'apeller plusieurs fois :

```
unsigned int fibo(unsigned int n) {
    if (n <= 2) return 1;
    else return fibo(n - 1) + fibo(n - 2);
}
```

Attention à l'explosion combinatoires ! ! !

1 La récursivité

2 Applications

3 En pratique

Principe général

La récursivité est un **outil algorithmique** très puissant. Il permet d'implémenter facilement les stratégieis "Diviser pour régner" :

Divide and conquer

- Diviser le problème à résoudre en un plusieurs problèmes plus simples.
- Résoudre récursivement le ou les problèmes plus simples.
- Fusionner les résultats de manière à obtenir le résultat final.

Cette stratégie est à la base de nombreux algorithmes que nous verrons plus tard.

Toujours plus loin !

Il y a de nombreuses façons de définir un problème plus simple.

Comment trier un paquet de cartes ?

- Séparer le paquet en deux demis-paquets
- Trier chaque paquet séparément
- Intercaler les cartes des deux paquets

Le tri des petits paquets peu se faire de la même manière jusqu'à ce que les sous paquets ne contiennent plus qu'une seule carte.

Toujours plus loin !

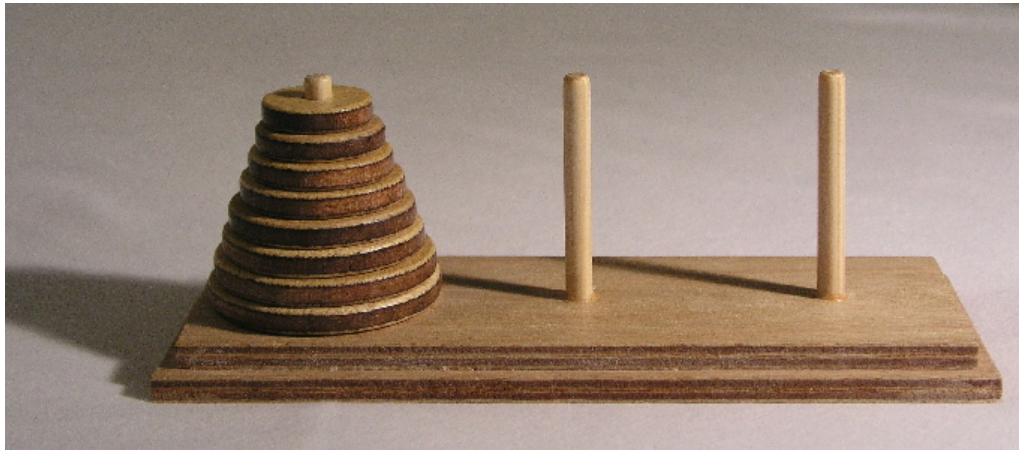
Il y a de nombreuses façons de définir un problème plus simple.

Comment trier un paquet de cartes ?

- Séparer le paquet en deux demis-paquets
- Trier chaque paquet séparément
- Intercaler les cartes des deux paquets

Le tri des petits paquets peu se faire de la même manière jusqu'à ce que les sous paquets ne contiennent plus qu'une seule carte.

Les tours de Hanoï



Comment déplacer les disques du premier au dernier pilier en :

- ne déplacement qu'un disque à la fois
- ne placant jamais un disque sur un autre disque plus petit ?

Les tours de Hanoï

Algorithme

Pour déplacer N disque du pilier 1 au pilier 3 :

- Si N=1, déplacer le disque ;
- Sinon :
  - Déplacer récursivement N-1 disques du pilier 1 vers le pilier 2 ;
  - Déplacer le disque restant du pilier 1 vers le pilier 3 ;
  - Déplacer récursivement N-1 disques du pilier 2 vers le pilier 3.

Équivalence avec les boucles

Définition

Équivalence La récursivité et la boucles ont la même expressivité.

En pratique, suivant les problèmes :

- une forme peut être plus simple à implémenter
- une forme peut être plus efficace

Équivalence avec les boucles

```
unsigned int fibo_rec(unsigned int n) {
    if (n <= 2) return 1;
    else return fibo_rec(n - 1) + fibo_rec(n - 2);
}
unsigned int fibo_iter(unsigned int n) {
        int a = 1, b = 1;
        if (n <= 2) return 1;
        while (n >= 2) {
                int t = a + b;
                a = b; b = t;
                n--;
        }
        return b;
}
```

1 La récursivité

2 Applications

3 En pratique

Exemple d'exécution d'une fonction récursive

`main` ligne 8, appel de `fact(5)`

```
 1  #include <iostream>
 2  using namespace std;
 3  int fact(int N) {
 4      if (N == 0) return 1;
 5      else return N*fact(N-1);
 6  }
 7  int main() {
 8      cout << fact(5) << endl;
 9      return 0;
10  }
```

Pile :

	7	?
	6	?
	5	?
	4	?
	3	?
	2	?
main	1	ADM
return	0	?

Exemple d'exécution d'une fonction récursive

Entrée de `fact`

```
 1  #include <iostream>
 2  using namespace std;
 3  int fact(int N) {
 4      if (N == 0) return 1;
 5      else return N*fact(N-1);
 6  }
 7  int main() {
 8      cout << fact(5) << endl;
 9      return 0;
10  }
```

	7	?
	6	?
	5	?
fact	4	**ADM**
N	3	**5**
return	2	?
	1	ADM
	0	?

Exemple d'exécution d'une fonction récursive

`fact` ligne 5 appel de `fact(4)` :

```
 1  #include <iostream>
 2  using namespace std;
 3  int fact(int N) {
 4      if (N == 0) return 1;
 5      else return N*fact(N-1);
 6  }
 7  int main() {
 8      cout << fact(5) << endl;
 9      return 0;
10  }
```

fact	7	**ADM**
N	6	**4**
return	5	?
	4	ADM
	3	5
	2	?
	1	ADM
	0	?

Exemple d'exécution d'une fonction récursive

`fact` ligne 5 appel de `fact(3)` :

```
 1  #include <iostream>
 2  using namespace std;
 3  int fact(int N) {
 4      if (N == 0) return 1;
 5      else return N*fact(N-1);
 6  }
 7  int main() {
 8      cout << fact(5) << endl;
 9      return 0;
10  }
```

fact	10	**ADM**
N	9	**3**
return	8	?
	7	ADM
	6	4
	5	?
	4	ADM
	3	5
	2	?
	1	ADM
	0	?

Exemple d'exécution d'une fonction récursive

`fact` ligne 5 appel de `fact(2)` :

```
1  #include <iostream>
2  using namespace std;
3  int fact(int N) {
4      if (N == 0) return 1;
5      else return N*fact(N-1);
6  }
7  int main() {
8      cout << fact(5) << endl;
9      return 0;
10 }
```

fact	13	**ADM**
N	12	**2**
return	11	?
	10	ADM
	9	3
	8	?
	7	ADM
	6	4
	5	?
	4	ADM
	3	5
	2	?
	1	ADM
	0	?

Exemple d'exécution d'une fonction récursive

`fact` ligne 5 appel de `fact(1)` :

```
 1  #include <iostream>
 2  using namespace std;
 3  int fact(int N) {
 4      if (N == 0) return 1;
 5      else return N*fact(N-1);
 6  }
 7  int main() {
 8      cout << fact(5) << endl;
 9      return 0;
10  }
```

fact	16	**ADM**
N	15	**1**
return	14	?
	13	ADM
	12	2
	11	?
	10	ADM
	9	3
	8	?
	7	ADM
	6	4
	5	?
	4	ADM
	3	5

Exemple d'exécution d'une fonction récursive

`fact` ligne 5 appel de `fact(0)` :

```
1  #include <iostream>
2  using namespace std;
3  int fact(int N) {
4      if (N == 0) return 1;
5      else return N*fact(N-1);
6  }
7  int main() {
8      cout << fact(5) << endl;
9      return 0;
10 }
```

fact	19	**ADM**
N	18	**0**
return	17	?
	16	ADM
	15	1
	14	?
	13	ADM
	12	2
	11	?
	10	ADM
	9	3
	8	?
	7	ADM
	6	4

Exemple d'exécution d'une fonction récursive

`fact` ligne 4 retour de 1 :

```
 1  #include <iostream>
 2  using namespace std;
 3  int fact(int N) {
 4      if (N == 0) return 1;
 5      else return N*fact(N-1);
 6  }
 7  int main() {
 8      cout << fact(5) << endl;
 9      return 0;
10  }
```

fact	19	ADM
N	18	0
return	17	**1**
	16	ADM
	15	1
	14	?
	13	ADM
	12	2
	11	?
	10	ADM
	9	3
	8	?
	7	ADM
	6	4

Exemple d'exécution d'une fonction récursive

`fact` ligne 5 retour de $1 * \texttt{fact}(0) = 1 * 1 = 1$:

```
 1  #include <iostream>
 2  using namespace std;
 3  int fact(int N) {
 4      if (N == 0) return 1;
 5      else return N*fact(N-1);
 6  }
 7  int main() {
 8      cout << fact(5) << endl;
 9      return 0;
10  }
```

	19	ADM
	18	0
	17	1
fact	16	ADM
N	15	1
return	14	**1**
	13	ADM
	12	2
	11	?
	10	ADM
	9	3
	8	?
	7	ADM
	6	4

Exemple d'exécution d'une fonction récursive

`fact` ligne 5 retour de $2 * \texttt{fact}(1) = 2 * 1 = 2$:

```
1  #include <iostream>
2  using namespace std;
3  int fact(int N) {
4      if (N == 0) return 1;
5      else return N*fact(N-1);
6  }
7  int main() {
8      cout << fact(5) << endl;
9      return 0;
10 }
```

	16	ADM
	15	1
	14	1
fact	13	ADM
N	12	2
return	11	**2**
	10	ADM
	9	3
	8	?
	7	ADM
	6	4
	5	?
	4	ADM
	3	5

Exemple d'exécution d'une fonction récursive

`fact` ligne 5 retour de $3 * \texttt{fact}(2) = 3 * 2 = 6$:

```
1  #include <iostream>
2  using namespace std;
3  int fact(int N) {
4      if (N == 0) return 1;
5      else return N*fact(N-1);
6  }
7  int main() {
8      cout << fact(5) << endl;
9      return 0;
10 }
```

	13	ADM
	12	2
	11	2
fact	10	ADM
N	9	3
return	8	**6**
	7	ADM
	6	4
	5	?
	4	ADM
	3	5
	2	?
	1	ADM
	0	?

Exemple d'exécution d'une fonction récursive

`fact` ligne 5 retour de $4 * \texttt{fact}(3) = 4 * 6 = 24$:

```
 1  #include <iostream>
 2  using namespace std;
 3  int fact(int N) {
 4      if (N == 0) return 1;
 5      else return N*fact(N-1);
 6  }
 7  int main() {
 8      cout << fact(5) << endl;
 9      return 0;
10  }
```

	10	ADM
	9	3
	8	6
fact	7	ADM
N	6	4
return	5	**24**
	4	ADM
	3	5
	2	?
	1	ADM
	0	?

Exemple d'exécution d'une fonction récursive

`fact` ligne 5 retour de $5 * \texttt{fact}(4) = 5 * 24 = 120$:

```
1  #include <iostream>
2  using namespace std;
3  int fact(int N) {
4      if (N == 0) return 1;
5      else return N*fact(N-1);
6  }
7  int main() {
8      cout << fact(5) << endl;
9      return 0;
10 }
```

	10	ADM
	9	3
	8	6
	7	ADM
	6	4
	5	24
fact	4	ADM
N	3	5
return	2	**120**
	1	ADM
	0	?

Exemple d'exécution d'une fonction récursive

`main` ligne 8 : affichage de `120`

```
 1  #include <iostream>
 2  using namespace std;
 3  int fact(int N) {
 4      if (N == 0) return 1;
 5      else return N*fact(N-1);
 6  }
 7  int main() {
 8      cout << fact(5) << endl;
 9      return 0;
10  }
```

	10	ADM
	9	3
	8	6
	7	ADM
	6	4
	5	24
	4	ADM
	3	5
	2	**120**
main	1	ADM
return	0	?

Exemple d'exécution d'une fonction récursive

retour du `main` ligne 9

```
 1  #include <iostream>
 2  using namespace std;
 3  int fact(int N) {
 4      if (N == 0) return 1;
 5      else return N*fact(N-1);
 6  }
 7  int main() {
 8      cout << fact(5) << endl;
 9      return 0;
10  }
```

	10	ADM
	9	3
	8	6
	7	ADM
	6	4
	5	24
	4	ADM
	3	5
	2	120
main	1	ADM
return	0	**0**

Fonction récursive avec référence

Toute fonction peut-être récursive, même avec des références !

Fonction récursive avec référence

`main` ligne 12, appel de `fact(3, res)`

```
 1  #include <iostream>
 2  using namespace std;
 3  void fact(int N, int &res) {
 4      if (N == 0) res = 1;
 5      else {
 6          fact(N-1, res);
 7          res = res * N;
 8      }
 9  }
10  int main() {
11      int res;
12      fact(3, res);
13      cout << res << endl;
14      return 0;
15  }
```

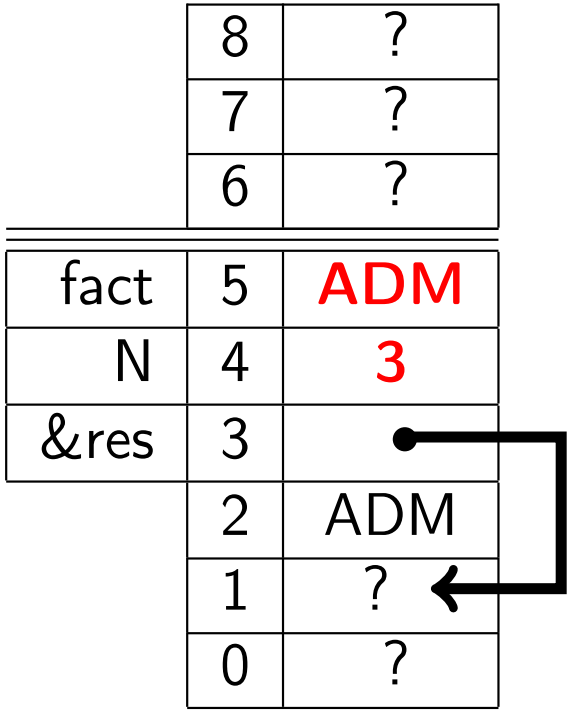
Pile :

	8	?
	7	?
	6	?
	5	?
	4	?
	3	?
main	2	ADM
res	1	?
return	0	?

Fonction récursive avec référence

Entrée de `fact`

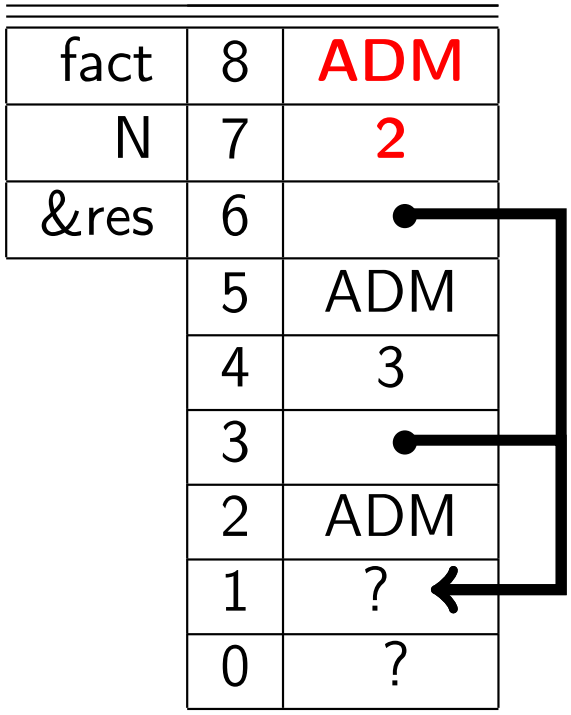
```
 1  #include <iostream>
 2  using namespace std;
 3  void fact(int N, int &res) {
 4      if (N == 0) res = 1;
 5      else {
 6          fact(N-1, res);
 7          res = res * N;
 8      }
 9  }
10  int main() {
11      int res;
12      fact(3, res);
13      cout << res << endl;
14      return 0;
15  }
```



Fonction récursive avec référence

`fact` ligne 6 appel de `fact(N-1, res)` :

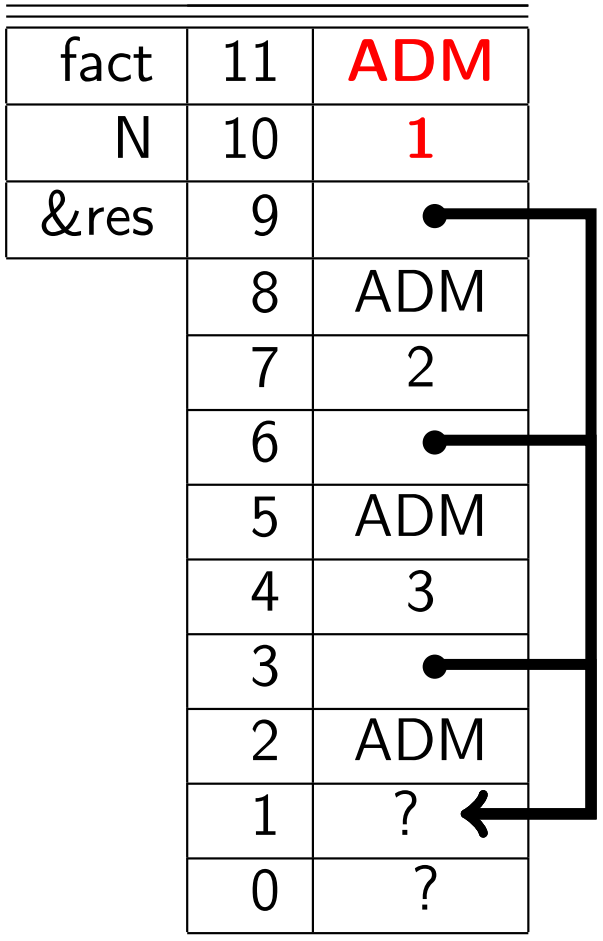
```
 1  #include <iostream>
 2  using namespace std;
 3  void fact(int N, int &res) {
 4      if (N == 0) res = 1;
 5      else {
 6          fact(N-1, res);
 7          res = res * N;
 8      }
 9  }
10  int main() {
11      int res;
12      fact(3, res);
13      cout << res << endl;
14      return 0;
15  }
```



Fonction récursive avec référence

`fact` ligne 6 appel de `fact(N-1, res)` :

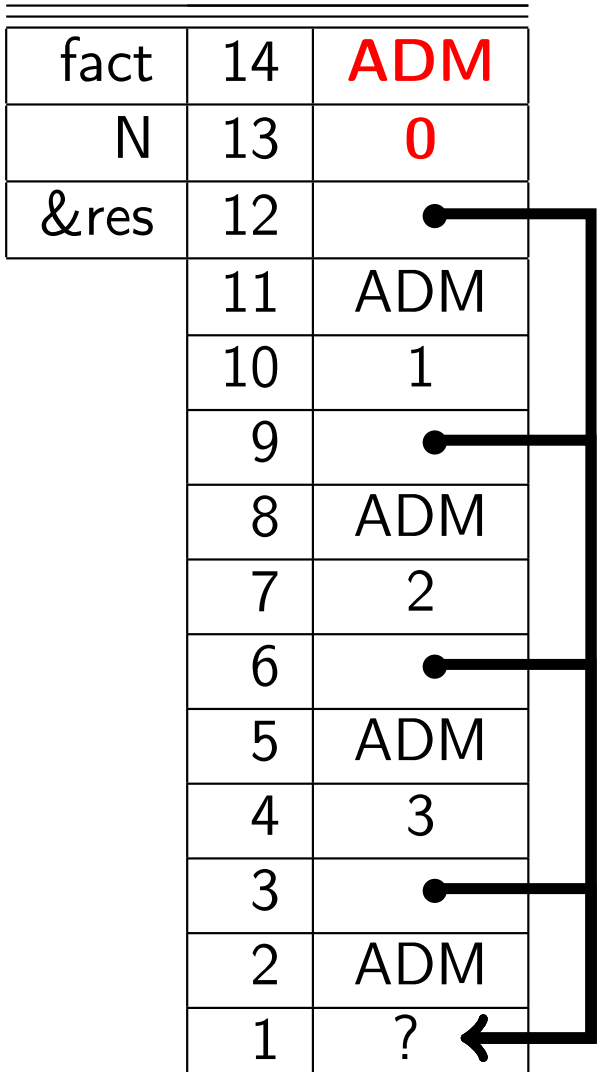
```
 1  #include <iostream>
 2  using namespace std;
 3  void fact(int N, int &res) {
 4      if (N == 0) res = 1;
 5      else {
 6          fact(N-1, res);
 7          res = res * N;
 8      }
 9  }
10  int main() {
11      int res;
12      fact(3, res);
13      cout << res << endl;
14      return 0;
15  }
```



Fonction récursive avec référence

Entrée de `fact(0, res)`

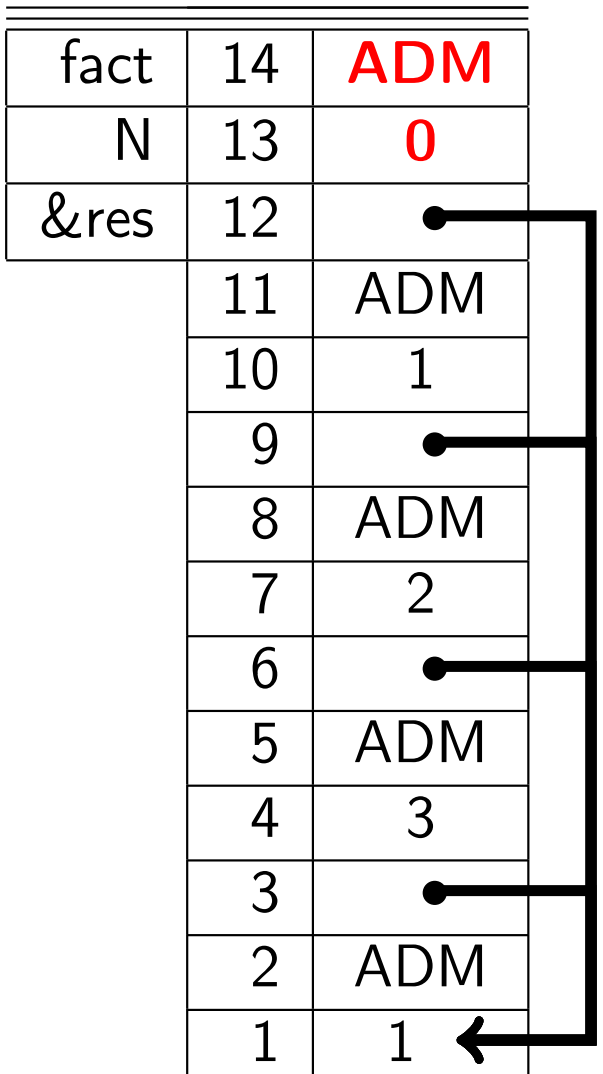
```
1  #include <iostream>
2  using namespace std;
3  void fact(int N, int &res) {
4      if (N == 0) res = 1;
5      else {
6          fact(N-1, res);
7          res = res * N;
8      }
9  }
10 int main() {
11     int res;
12     fact(3, res);
13     cout << res << endl;
14     return 0;
15 }
```



Fonction récursive avec référence

Ligne 4, calcul du résultat pour N = 0

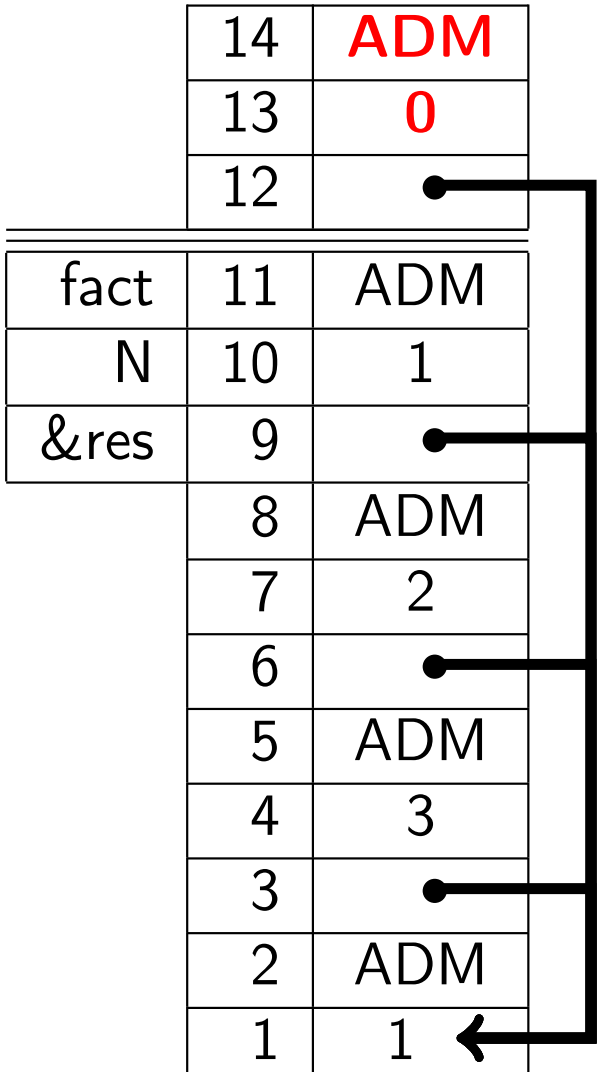
```
1  #include <iostream>
2  using namespace std;
3  void fact(int N, int &res) {
4      if (N == 0) res = 1;
5      else {
6          fact(N-1, res);
7          res = res * N;
8      }
9  }
10 int main() {
11     int res;
12     fact(3, res);
13     cout << res << endl;
14     return 0;
15 }
```



Fonction récursive avec référence

Ligne 7, calcul du résultat pour N = 1

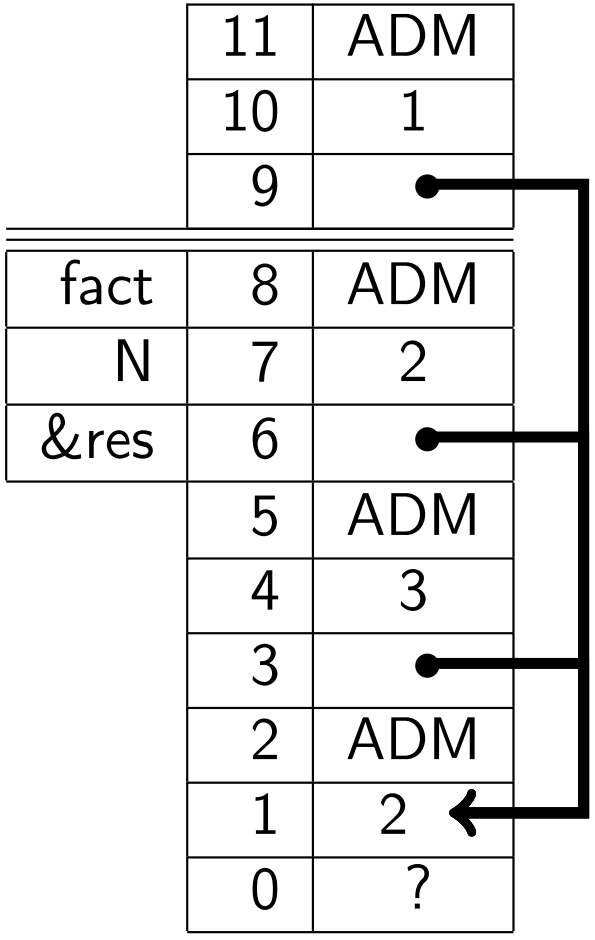
```
1  #include <iostream>
2  using namespace std;
3  void fact(int N, int &res) {
4      if (N == 0) res = 1;
5      else {
6          fact(N-1, res);
7          res = res * N;
8      }
9  }
10 int main() {
11     int res;
12     fact(3, res);
13     cout << res << endl;
14     return 0;
15 }
```



Fonction récursive avec référence

Ligne 7, calcul du résultat pour N = 2

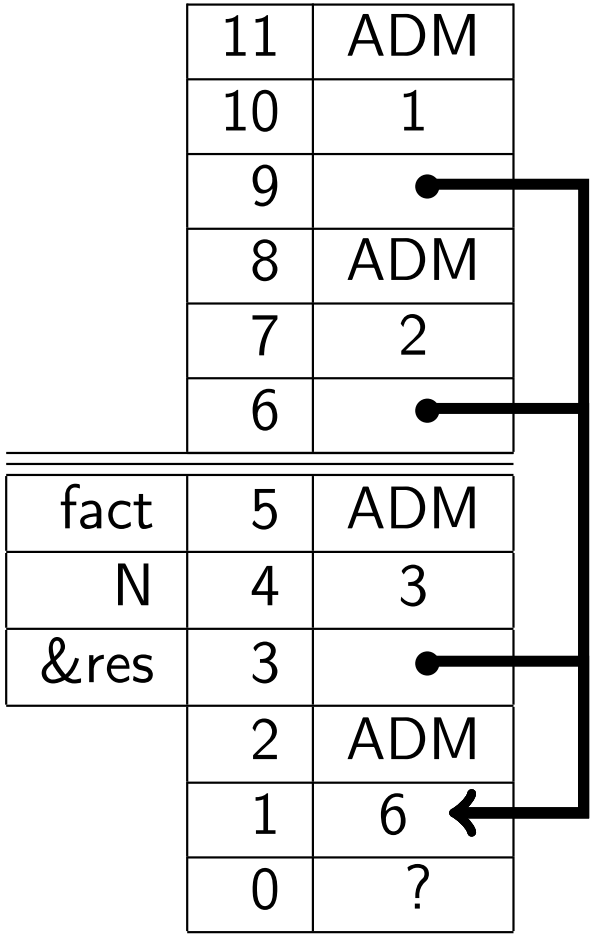
```
1  #include <iostream>
2  using namespace std;
3  void fact(int N, int &res) {
4      if (N == 0) res = 1;
5      else {
6          fact(N-1, res);
7          res = res * N;
8      }
9  }
10 int main() {
11     int res;
12     fact(3, res);
13     cout << res << endl;
14     return 0;
15 }
```



Fonction récursive avec référence

Ligne 7, calcul du résultat pour N = 3

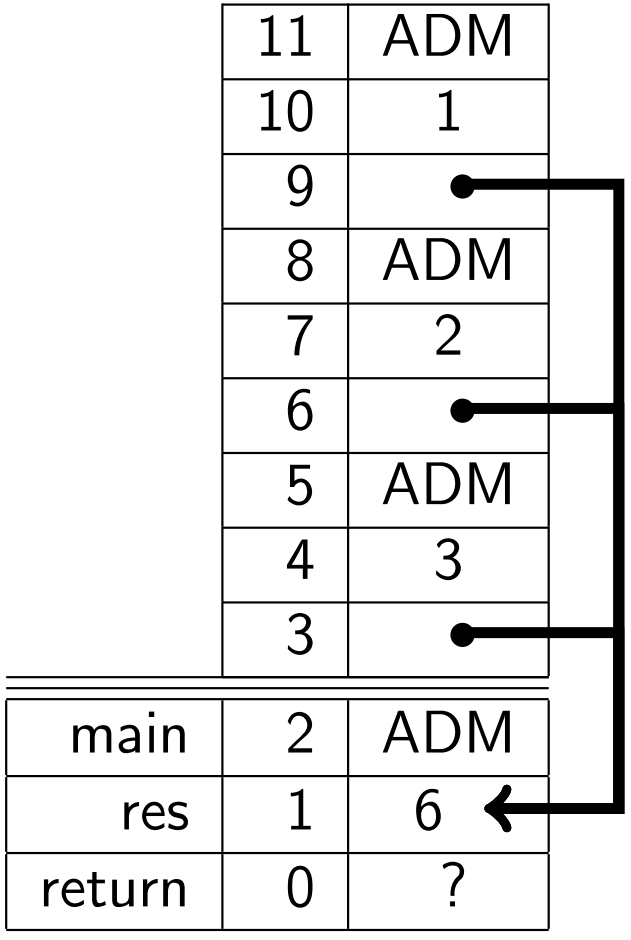
```
1  #include <iostream>
2  using namespace std;
3  void fact(int N, int &res) {
4      if (N == 0) res = 1;
5      else {
6          fact(N-1, res);
7          res = res * N;
8      }
9  }
10 int main() {
11     int res;
12     fact(3, res);
13     cout << res << endl;
14     return 0;
15 }
```



Fonction récursive avec référence

Affichage du résultat

```
 1  #include <iostream>
 2  using namespace std;
 3  void fact(int N, int &res) {
 4      if (N == 0) res = 1;
 5      else {
 6          fact(N-1, res);
 7          res = res * N;
 8      }
 9  }
10  int main() {
11      int res;
12      fact(3, res);
13      cout << res << endl;
14      return 0;
15  }
```



Récursion et Pile

Retenir

Chaque appel récursif consomme de la mémoire dans la pile !

Attention ! La pile a une taille fixée au départ du programme, une mauvaise utilisation de la récursivité peut entraîner un débordement de pile.
Le système augmente automatiquement la taille de la pile mais dans certaines limites seulement !

C'est le fameux **Stack Overflow** !

Récursion et Pile

Retenir

Chaque appel récursif consomme de la mémoire dans la pile !

Attention ! *La pile a une taille fixée au départ du programme, une mauvaise utilisation de la récursivité peut entraîner un débordement de pile.*
Le système augmente automatiquement la taille de la pile mais dans certaines limites seulement !

C'est le fameux **Stack Overflow** !