CHAP IV LA RECURSIVITE

I Introduction

La récursivité peut s'appliquer à un objet ou à une fonction. Un objet est récursif lorsque sa définition fait référence â lui-même. Une fonction est récursive lorsqu'on fait appel à la fonction.

Exemples : $n! = n*(n-1)!$ $x^n = x*x^{n-1}$

récursivité --> Données --> Types récursifs

-->fonctions --> Programmation récursive

Langages de programmation --> Pascal, C,C++,JAVA ...

non récursifs --> Basic, Fortran, ...

On introduit la notation récursive en étudiant des problèmes (ou des programmes) dont l'objet est de calculer la valeur des fonctions définies par récurrence. Leur construction est fondée sur une analyse récurrente : On définit une définition récurrente de la fonction puis l'on construit le programme (ou l'arbre) récursif (Récurrence implicite ou explicite par une formule) .

Les langages PASCAL, C, JAVA, ... autorisent l'utilisation récursive des sous-programmes.

Une fonction est utilisée **récursivement** si le nom de cette fonction apparaît dans une expression de la partie instruction du bloc ou d'un bloc inclus.

Trois phases dans la conception d'un algorithme récursif :

a) spécification des objets = paramètres formels :

La valeur de l'un de ces paramètres doit croître ou décroître a chaque appel récursif ...

b) recherche de la (ou des) condition(s) d'arrêt :

Situation n'exigeant aucun appel récursif :"cas trivial" où la la taille du problème est nulle ou 1.

c) généralisation de la solution

On décompose le problème en cas trivial et sous-problèmes de même nature mais simples ...

NB: b) et c) reviennent à rechercher la récurrence!
D'où:
* Analyse récurrente : formule de récurrence ou récurrence
* puis notation récursive = solution récursive
* enfin exécution d'un algorithme récursif

II ANALYSE RECURRENTE

TROUVER UNE RÉCURRENCE OU UNE FORMULE DE RÉCURRENCE

II-1 EXEMPLES NUMÉRIQUES

n!=> $F_n = n*F_{n-1}$ $F_0 = 1$

f(n) = n * f(n-1) (fonction mathématique récursive)

=> Récurrence => appels récursifs

Conditions initiales => Critère d'arrêt

Autres exemples mathématiques avec les suites récurrentes

$U_n = f(U_{n-1})$ fonction U(n) = f (U(n-1))

$U_0 = a$ U(0) = a

$U_n = f(U_{n-1}, U_{n-2})$ U(n) = f (U(n-1), U(n-2))

$U_0 = a$ $U_1 = b$ U(0) = a U(1) = b

$S_k = S_{k-1} + T_k$, $S_0 = T_0$ S(n)=S(n-1)+T(n), S(0)=T(0)

II-2 EXEMPLES SUR DES TABLEAUX

typedef int Tab [Max] ;

On veut définir une fonction SUM qui calcule la somme des (k+1) premières valeurs (rangées dans les cases de 0 à k) d'un tableau appelé T de type Tab

Cas Particulier : **sum (T, 0) = T [0]**

Récurrence : **sum (T, k) = T [k] + sum (T, k-1)**

Autre solution :

Cas particulier : **sum (T, k+1) = 0**

Récurrence : **sum (T, k) =T [k] + sum (T, k+1)**

Autre exemple : On veut définir une fonction MIN qui trouve la valeur minimale des (k +1)^{èmes} premières valeurs de T de type Tab.

Cas Particulier **MIN (T,0) = T [0]**

Récurrence: **MIN (T,k)= Mini (T[k], MIN(T,k-1))**

où mini est ici une simple fonction qui retourne la valeur minimale de 2 valeurs de même type.

Ce serait pareil si T était un champ d'une variable S de type structure :

Cas Particulier **MIN (S,0) = S.T [0]**

Récurrence:**MIN(S,k)=Mini (S.T[k], MIN(S, k-1))**

II-3 Exemples sur les chaînes

- On veut définir la fonction LG - longueur de chaînes - qui associe â une chaîne de caractères sa longueur.

LG

L* ----------> N

Condition initiale : **LG (' \0 ') = 0**

Récurrence : **LG (c.l) = 1+LG (c)**

cette relation exprime le fait que si l'on rajoute un caractère, sa longueur augmente de 1.

Autre exemple avec l'égalité de 2 chaînes où l'on veut définir la fonction qui associe à deux chaînes une valeur logique / booléenne

VRAI si = FAUX si non =

EG

L* . L* ----------> {VRAI, FAUX}

Cas particulier et condition initiale :

EG (' ‘ , ' ') = VRAI

EG (' ' , c.l) = FAUX

EG (c.l , ' ') = FAUX

Récurrence : **EG (c1.l1, c2.l2) = [(l1 = l2) et EG (c1,c2)]**

ces chaînes ont au moins un caractère

c1, c2	L*
l1, l2	L

II NOTATION RECURSIVE

Un algorithme récursif se représente par une fonction c'est à dire tel que dans sa définition, on fait appel à elle-même (nom de la fonction).

condition initiale => condition d'arrêt

récurrence => appel récursif

Forme générale

si condition_arret

alors action

sinon actions dont au moins un appel récursif

si non condition_arret

alors actions dont au moins un appel récursif

III-1 EXEMPLE 1 : N!

$F_0 = 1$

$F_n = n * F_{n-1}$ pour n>=1

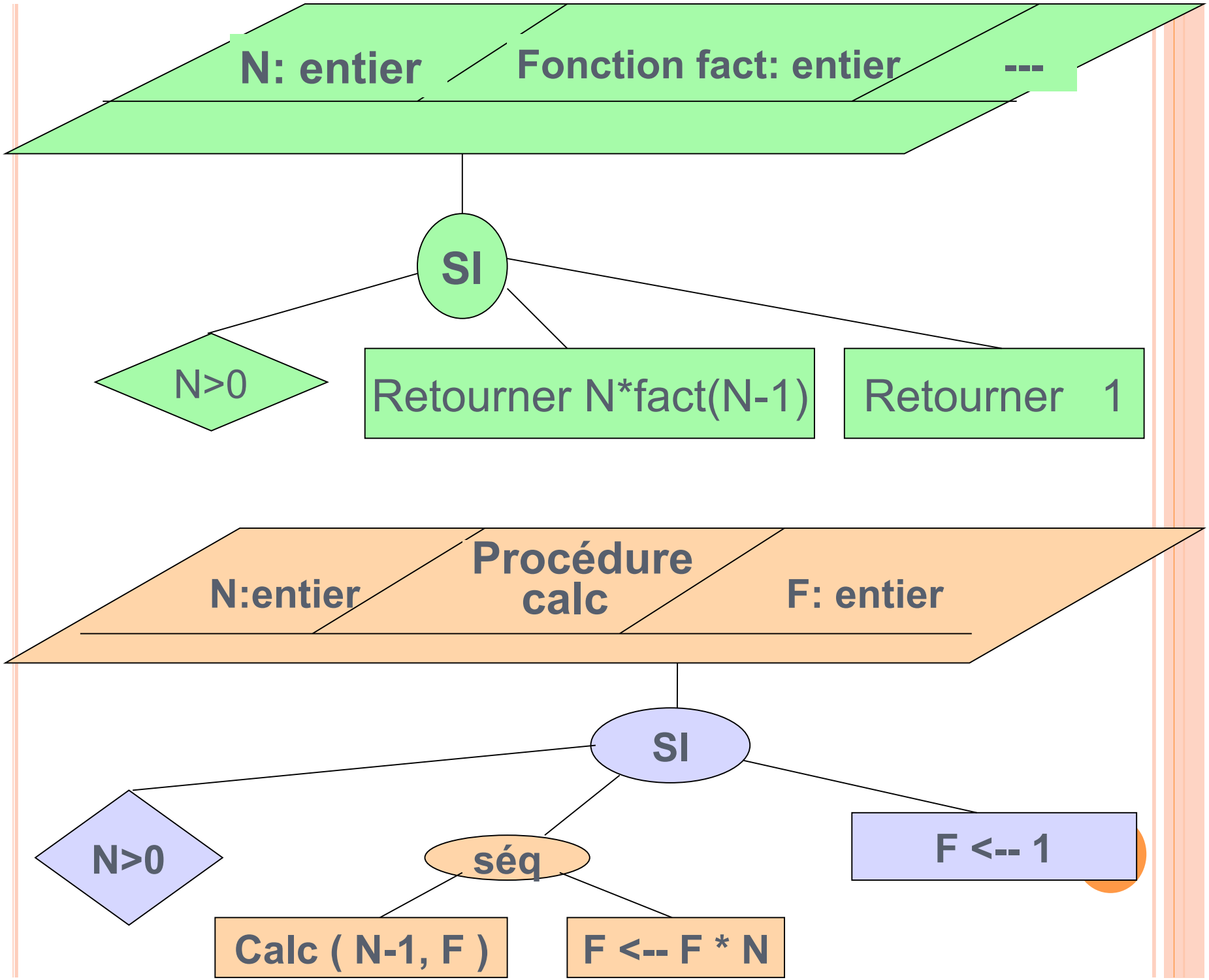
a) n! => ***int fact (int n)***

b) Condition d'arrêt n = 0 => ***return 1***

c) Cas général = (récurrence) ***return n * fact (n-1)***

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

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


N: entier
Fonction fact: entier

SI
N>0
Retourner N*fact(N-1)
Retourner 1
N:entier
Procédure calc
F: entier
SI
N>0
séq
F <-- 1
Calc (N-1, F)
F <-- F * N

III-2 Exemple 2 : Suite de Fibonnaci

$U_0 = 0$ $U_1 = 1$ $U_n = U_{n-1} + U_{n-2}$ pour $n > 1$

0 1 1 2 3 5 8 13 21 34 55 89 144 233

1) U_n => ***int fib (int n)***

2) Stop condition n = 0 ou 1 => ***return 0 ou 1***

3) Cas général ***return fib (n-1) + fib (n-2)***

III-3 Exemple 3 : le triangle de Pascal

Calcul des coefficients du Binôme de Newton,

a) spécification : en entrée n, p : entier

et en sortie: entier ***int cnp (int n, int p)***

b) Condition d'arrêt n=0 ou p=0 => ***return 1***

c) Cas général ***return cnp (n-1, p-1) + cnp (n-1, p)***

III-4 Exemple 4 : Longueur de chaine

1) Spécification des paramètres

entrée => s: Chaine, t : index de la chaîne (entier),

sortie : entier (longueur).

int lg (Chaine s , int t)

2) Stop_condition s[t] = '\0 ' => *return t*

3) Cas général *return lg (s, t +1)*

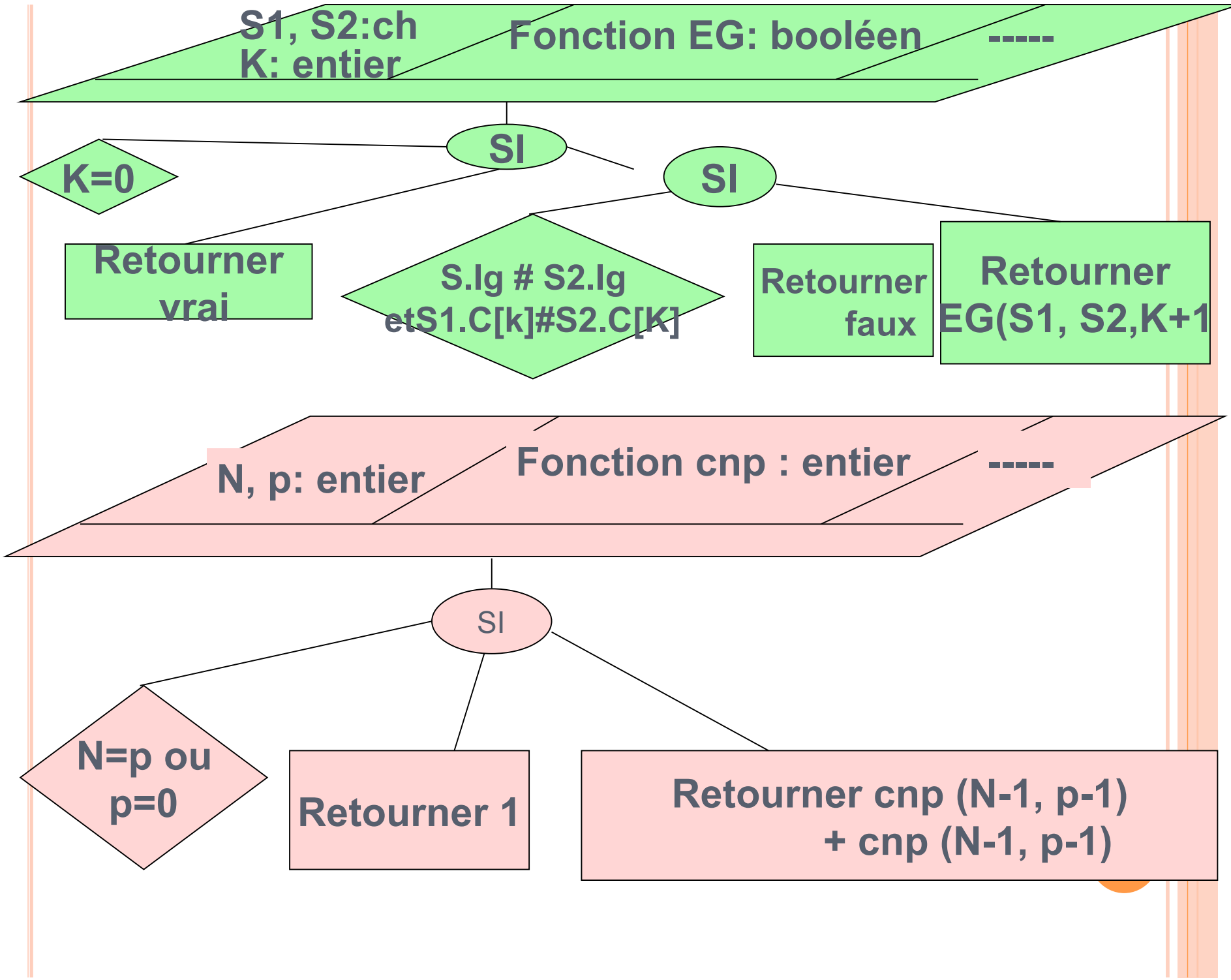
Appel de la fonction *lg (s, 0)*

```
int lg ( Chaine s, int t )
{ if (s[ t ] == '\0') return  t;
    else return lg ( s , t + 1 );}
```

III-5 Exemple 5 : Egalité de 2 chaînes

```
typedef  struct      {   short lg ;
                         char c [MXCH ];
                     }  Ch;
```

1) en entrée s1, s2 : Ch ; k : entier
   en sortie eg : booleen (entier)
   int eg (Ch s1, Ch s2 , int k)
2) s1.lg # s2.lg=> *return faux (0)*
   s1.c[k] # s2.c[k] => *return faux (0)*
   k = s1.lg => *return vrai (1)*
3) *return eg (s1, s2, k + 1)*


S1, S2:ch
K: entier
Fonction EG: booléen

SI
K=0
SI
Retourner vrai
S.lg # S2.lg etS1.C[k]#S2.C[K]
Retourner faux
Retourner EG(S1, S2,K+1
N, p: entier
Fonction cnp : entier

SI
N=p ou p=0
Retourner 1
Retourner cnp (N-1, p-1) + cnp (N-1, p-1)

III EXECUTION D'ALGORITHMES RECURSIFS

Lors de l'exécution d'un algorithme (fonction) récursif, l'algorithme s'appelle lui-même tant que la condition d'arrêt n'est pas vérifiée.

Les paramètres et les variables locales de la fonction sont empilés à chaque appel récursif de la fonction (Interruption de la fonction). Lorsque la fonction appelée se termine, il y a alors retour à celle qui l'a appelée: on récupère les paramètres et les variables locales qui sont dépilés et la fonction interrompue reprend (continue) à l'instruction qui suit l'appel récursif.

III.1 Exemple de Factorielle n 4!

appel de FACT(4) c-à-d 4! --> 3! --> 2! --> 1! ->0!

FACT(3)

FACT(2)

FACT(1)

FACT(0)

FACT(0) = 1

puis remontée

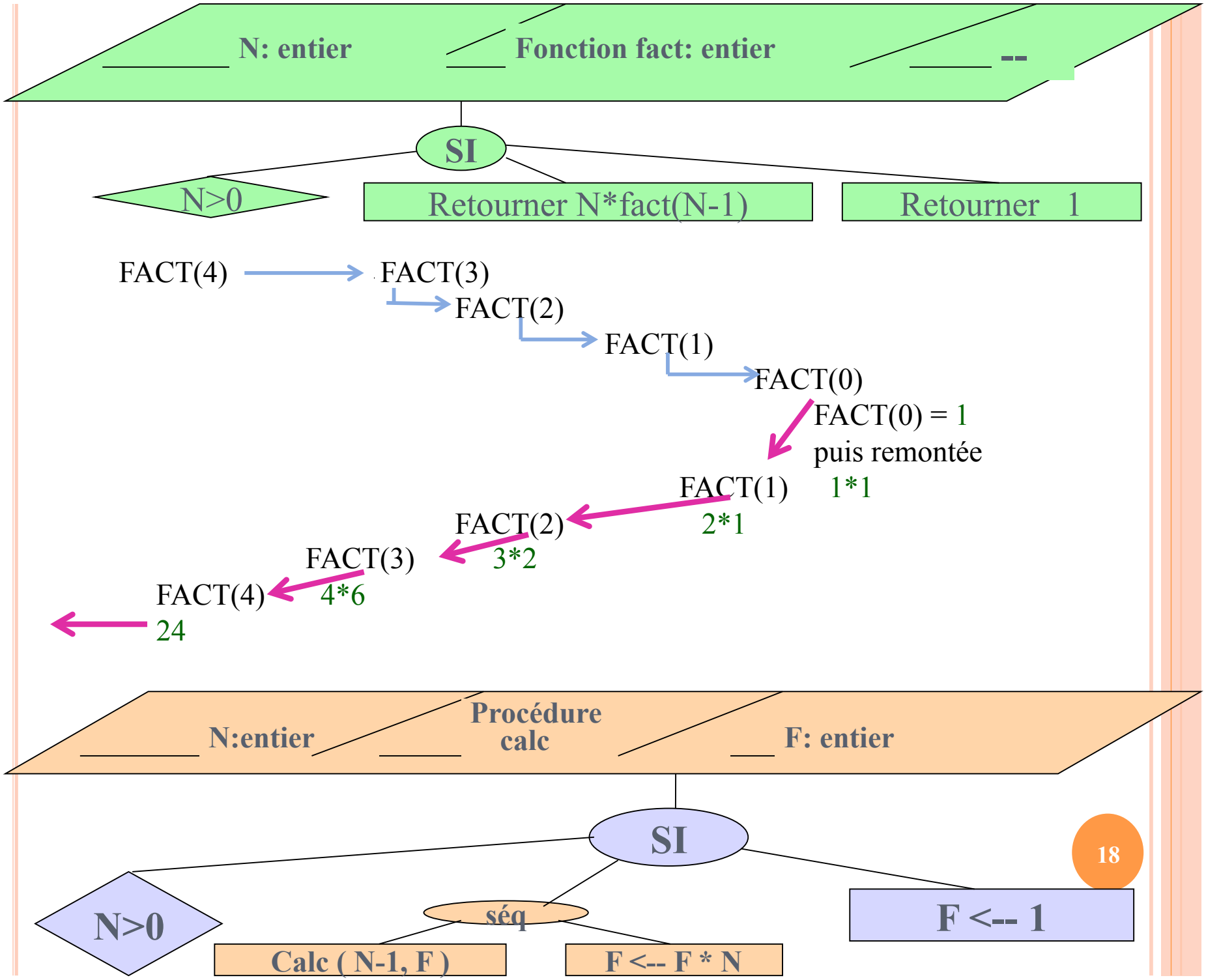
FACT(1) 1*1

FACT(2) 2*1

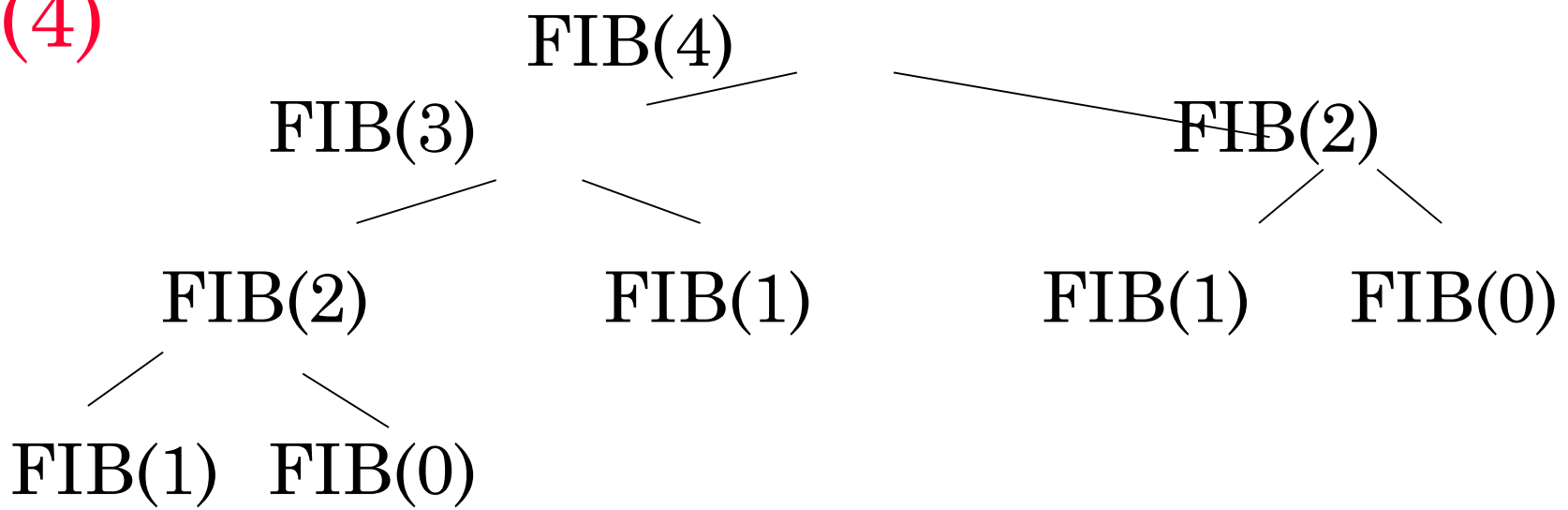
FACT(3) 3*2

FACT(4) 4*6

24


N: entier
Fonction fact: entier
--
SI
N>0
Retourner N*fact(N-1)
Retourner 1
FACT(4)
FACT(3)
FACT(2)
FACT(1)
FACT(0)
FACT(0) = 1
puis remontée
FACT(1)
1*1
FACT(2)
2*1
FACT(3)
3*2
FACT(4)
4*6
24
N:entier
Procédure
calc
F: entier
SI
N>0
séq
Calc (N-1, F)
F <-- F * N
F <-- 1

III.2 Appels récursifs de FIB pour FIB (4)



Le paramètre n est passé par valeur et donc un appel de fonction consiste à l'évaluer puis à se lancer dans l'exécution de la fonction avec la valeur de l'argument.

fib (4) = fib (3) + fib (2)
fib (3) = fib (2) + fib (1)
fib (2) = fib (1) + fib (0)

IV RECURRENCE IMPLICITE ET NOTATION RECURSIVE

L'analyse récurrente peut faire apparaître une récurrence implicite qui ne s'exprime pas par une formule de récurrence. La récurrence traduit le fait qu'on est confronté au même problème mais avec moins de données :

- soit on décrit les mêmes instructions sur un sous-ensemble de valeurs : n-1 ou ....

- soit il s'agit d'une récurrence graphique : dessin, ...

Exemple sur la saisie d'un tableau de n valeurs

```
void saisie ( int  t [ ] , int n)
  { if ( n>0) {  scanf ( «%d», & t [ n-1]) ;
                 saisie ( t , n-1 ) ; }
  }
```

```
void saisie (int  t [ ] , int n)
     { if ( n>0) { saisie ( t , n-1 ) ;
                   scanf ( «%d», & t [ n-1]) ; }
     }
```

```
void saisie (int  * pt  , int n)
     { if ( n>0) { saisie ( pt , n-1 ) ;
                   scanf ( «%d»,  (pt + n-1)  ) ; }
     }
```

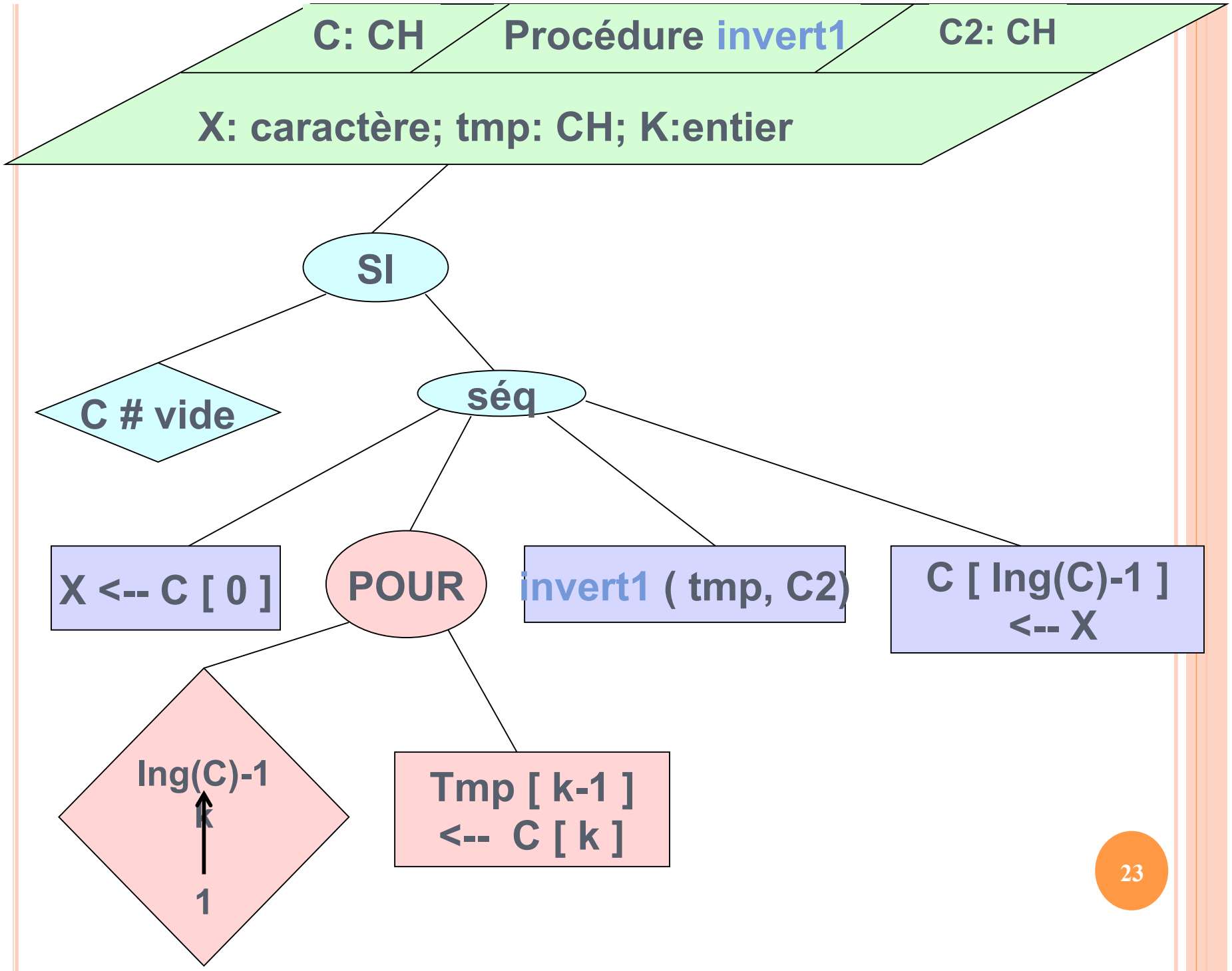
IV.1 Un exemple : Inversion de mots

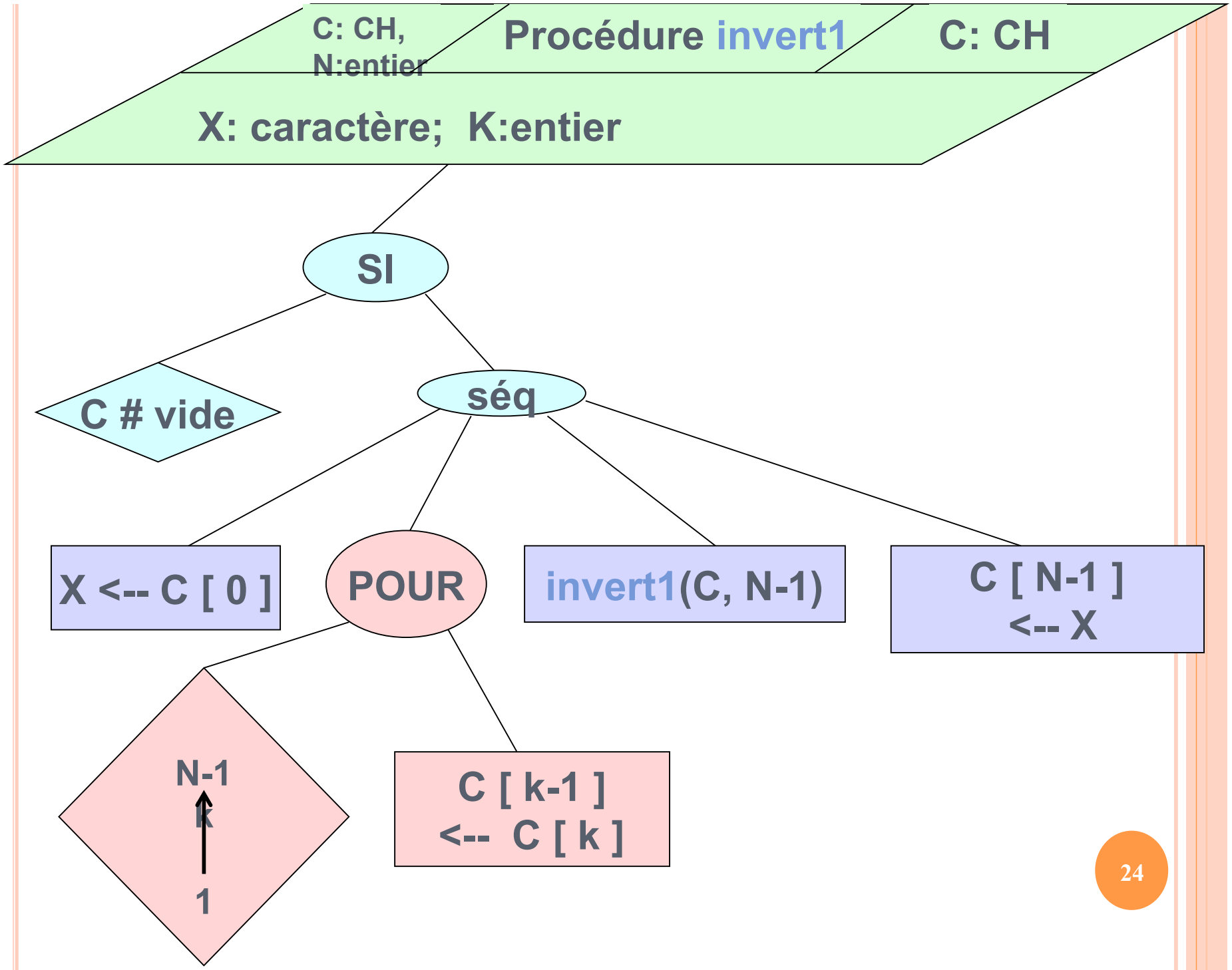
On veut inverser l'ordre des lettres d'un mot (‘noel’ en ‘leon’).
Ce problème peut-être résolu en ôtant la 1ère lettre du mot, en inversant le reste du mot et en ajoutant ensuite la lettre ôtée

inverser ***(mot)***

ôter la 1ère lettre;
inverser ***le reste du mot****;*
ajouter la lettre ôtée.

Ceci est un algorithme récursif (mais pas de relation de récurrence), le cas limite correspond à une seule lettre ou 0 lettre (vide).


C: CH
Procédure invert1
C2: CH
X: caractère; tmp: CH; K:entier
SI
C # vide
séq
X <-- C [0]
POUR
invert1 (tmp, C2)
C [lng(C)-1]
<-- X
lng(C)-1
k
1
Tmp [k-1]
<-- C [k]


C: CH, N:entier
Procédure invert1
C: CH
X: caractère; K:entier
SI
C # vide
séq
X <-- C [0]
POUR
invert1(C, N-1)
C [N-1] <-- X
N-1
k
1
C [k-1] <-- C [k]

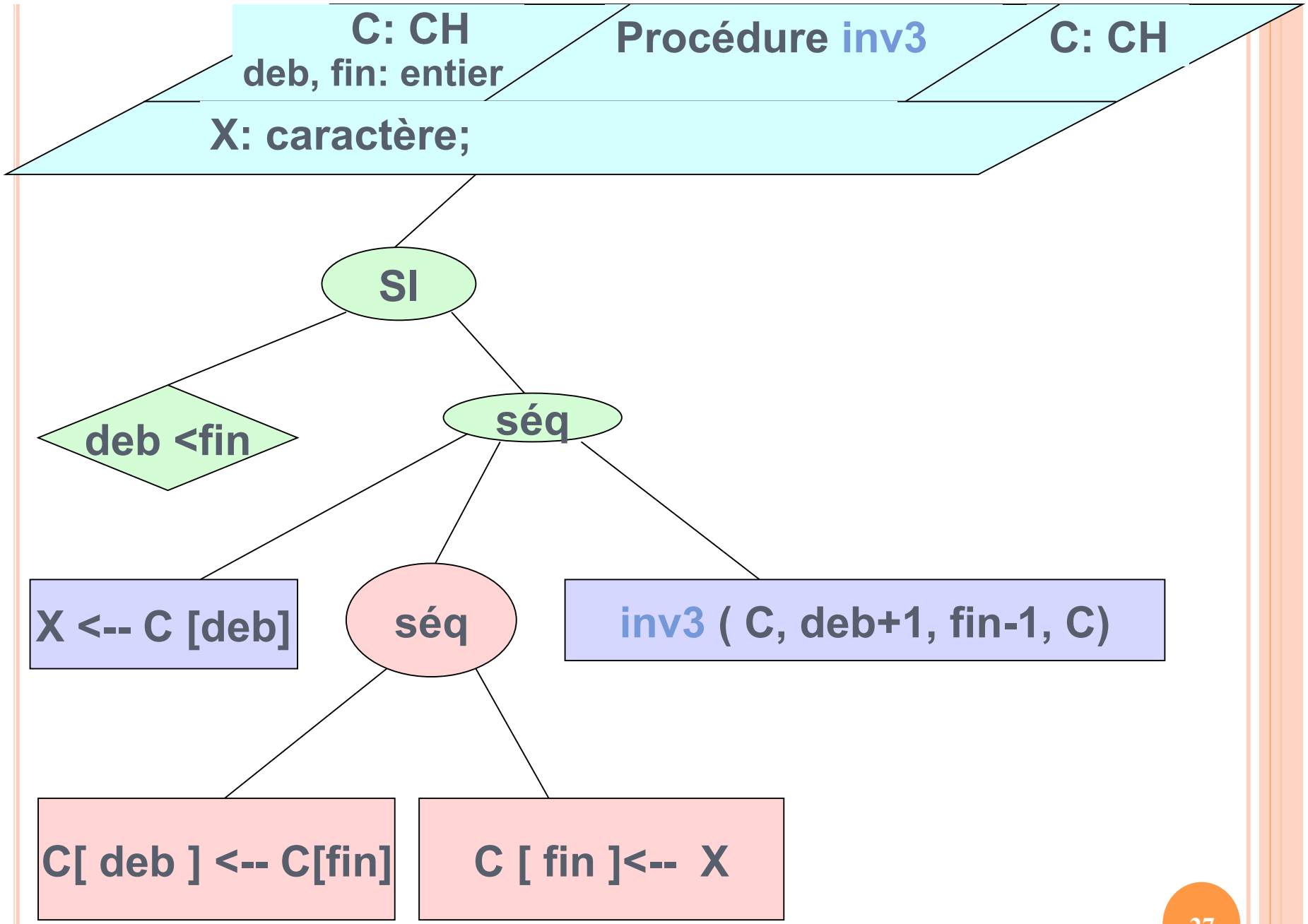
```
void   inverser ( char ch [ ] , int n )
   {   char c;   int  i = 0 ;
        if  ( n>0 ){   c = ch [ 0 ] ;
                    while (++i < n )  ch [i-1] = ch[i];
                    inverser ( ch, n-1 ) ;
                    ch [n-1] = c;
                  }
   }
```

```
void   inverser ( char * pch , int n )
{     char c;   int  i = 0 ;   if  ( n>0 )
        { c = * pch ;
          while (++i < n )  *(pch+i-1)= *(pch+i);
          inverser ( pch , n-1 ) ;
          *(pch +n-1) = c;
        }
}
```

BIEN DÉFINIR LA MÉTHODE ET L'EN-TÊTE DU SOUS-PROGRAMME (PRÉCISER CE SUR QUOI VA PORTER L'APPEL RÉCURSIF ...)

UNE AUTRE SOLUTION EST DE FAIRE APPARAÎTRE 2 PARAMÈTRES SUPPLÉMENTAIRES *DEBUT* ET *FIN*: ENTIER .

```
void inv (char c [  ], int deb, int fin )
    { char x ;
      if  ( deb < fin)
         {   x = c [deb];
             c [deb] = c[ fin]; c[fin] = x;
             inv ( c, deb+1, fin-1 );
         }
    }
```

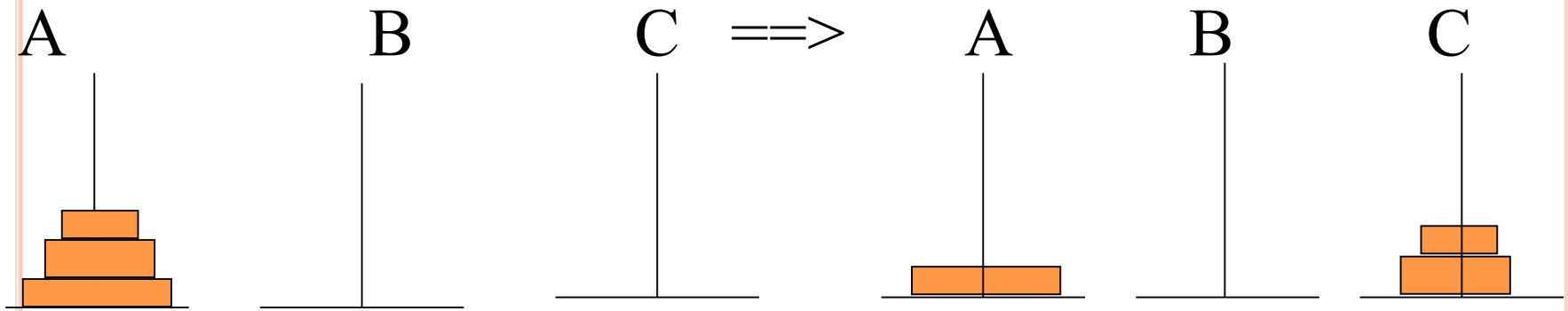


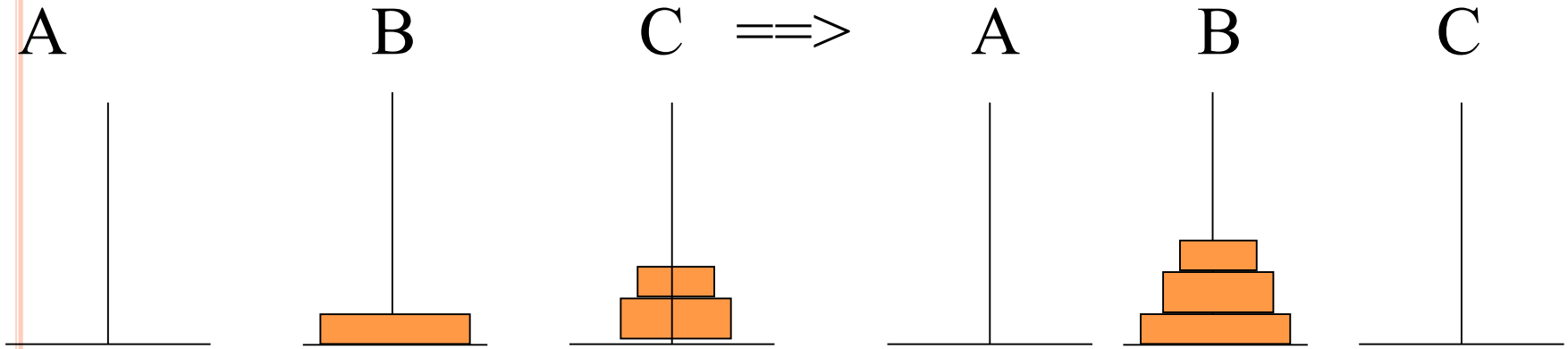
Récursivité terminale

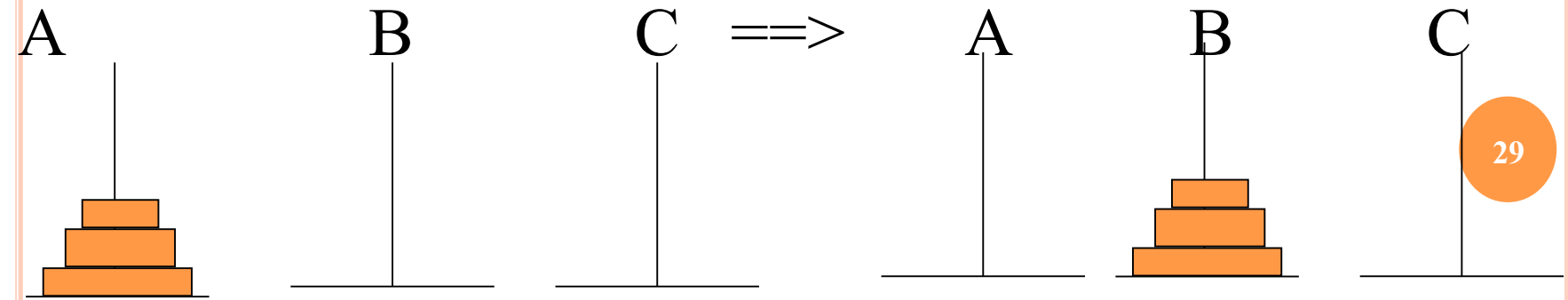
IV.2 Problème des Tours de HANOI

- On dispose de 3 piquets ou socles et d'un ensemble de N disques qui au départ sont placés par ordre décroissant sur l'un des socles. Le but du jeu est de placer progressivement tous ces disques sur un autre socle en ne déplaçant qu'un disque à la fois et en respectant la règle de ne jamais placer un disque sur un disque de diamètre inférieur.
- Ce problème, très difficile en apparence, se résout très simplement quand on raisonne par récurrence.

A B C ==> A B C


A
B
C
==>
A
B
C


A
B
C
==>
A
B
C


A
B
C
==>
A
B
C

OBJECTIF : ECRIRE UN SOUS-PROGRAMME QUI INDIQUE LA SUITE DES TRANSFERTS À EFFECTUER POUR N QUELCONQUE.

La solution générale est décrite par une récurrence graphique qui peut alors s'exprimer par la notation récursive suivante:

TRANSFERER N disques du support 1 vers le support 2:

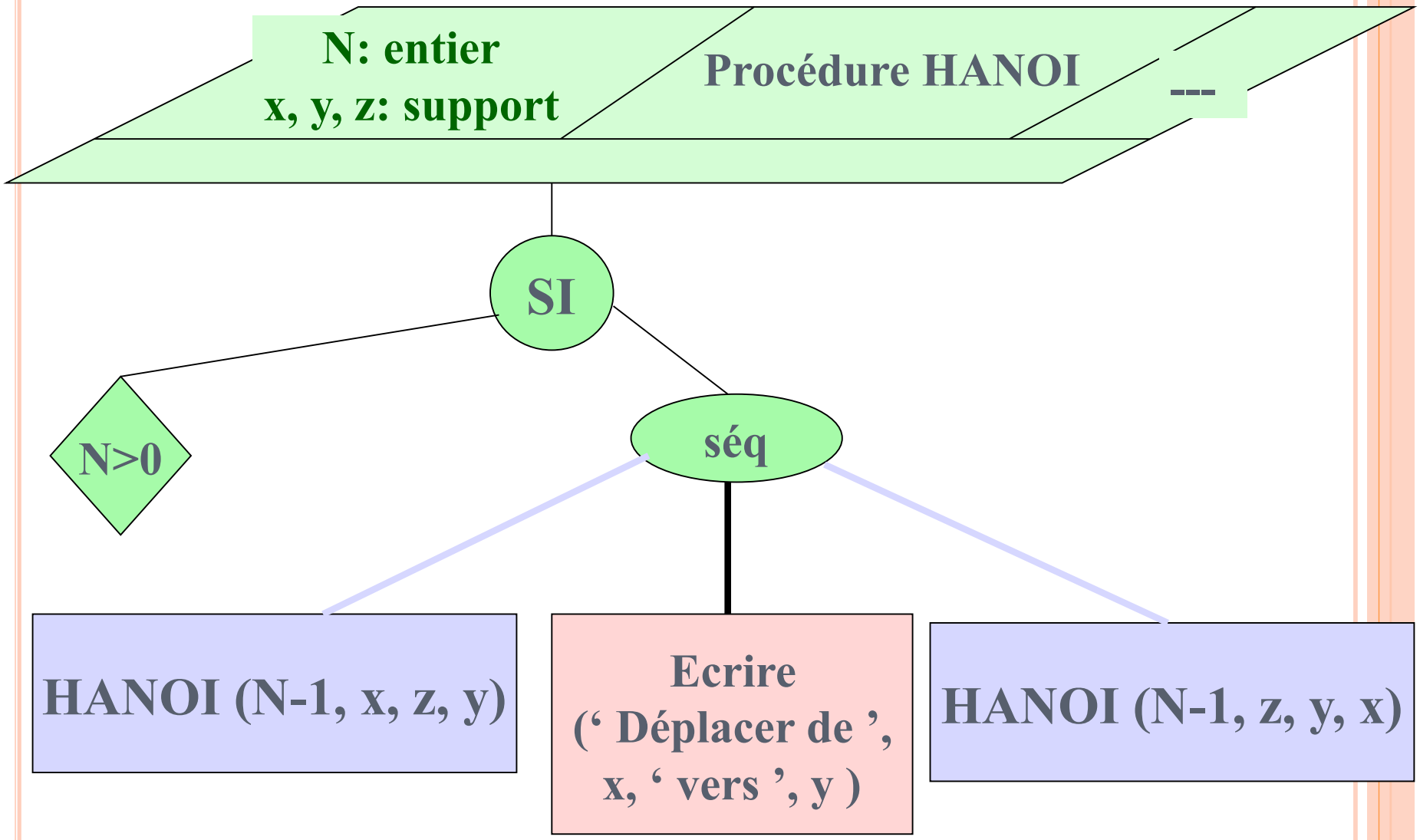
1)**TRANSFERER** N-1 disques du support 1 vers 3;

2) Déplacer le disque du support 1 vers le support 2 ;

3)**TRANSFERER** N-1 disques du support 3 vers 2;

Le cas particulier correspond au cas où aucun disque n'est sur le support : il n'y a alors rien à faire: N=0 => rien!

TRANSFERER N disques du support 1 vers le support 2:
pour cela on a besoin d 'un support 3 intermédiaire.
D'où 4 paramètres en entrée notés N:entier et x, y, z:support



Paramétrage

n : entier (nombre de disques)

x, y, z: socle (caractère ou entier ou énuméré)

```
typedef  char  Socle ;
void hanoi ( int n , Socle x , Socle y , Socle z  )
    {   if ( n>0 )
        {   hanoi ( n-1 , x, z ,y ) ;
            printf («Déplacer de %c vers %c\n», x,y) ;
            hanoi ( n-1 , z , y , x ) ;
        }           }
```

```
main ( )
        {   int nb ; scanf ( « %d »  , &nb ) ;
            hanoi ( nb , 'A', 'B', 'C') ;      }
```

hanoi n= 3 de ‘A’ vers ‘B’ en utilisant ‘C’

Appel de hanoi (2, ‘A’, ‘C’, ‘B’)

Appel de hanoi (1, ‘A’, ‘B’, ‘C’)

Appel de hanoi (0, ‘A’, ‘C’, ‘B’)

Déplacer de A vers B

Appel de hanoi (0, ‘C’, ‘B’, ‘A’)

Déplacer de A vers C

Appel de hanoi (1, ‘B’, ‘C’, ‘A’)

Appel de hanoi (0, ‘B’, ‘A’, ‘C’)

Déplacer de B vers C

Appel de hanoi (0, ‘A’, ‘C’, ‘B’)

Déplacer de A vers B

Appel de hanoi (2, ‘C’, ‘B’, ‘A’) etc...

Flocon de von koch par exemple :

- Flocon d'ordre 0 est un triangle équilatéral
- Flocon d'ordre 1 est ce même triangle dont les côtés sont découpés en 3 et sur lequel s'appuie un autre triangle equilatéral au milieu
- flocon d'ordre n + 1 consiste à prendre le flocon d'ordre n en appliquant la même opération sur chacun des côtés

```
void saisie (int  * pt  , int n)
    { if ( n>0) {   saisie ( pt , n-1 ) ;
                    scanf ( «%d»,  (pt + n-1)  ) ;
                }
    }
```

```
void saisie ( int  * pt  , int n)
    { if ( n>0) {   saisie ( pt + 1 , n-1 ) ;
                    scanf ( «%d»,  pt   ) ;
                }
    }
```

```
void saisie (int  * pt  , int n)
    { if ( n>0) {   scanf ( «%d»,  pt   ) ;
                    saisie ( pt + 1 , n-1 ) ;
                }
    }
```