

# STRUCTURES DE DONNÉES

## MIP — S4

---

Pr. K. Abbad & Pr. Ad. Ben Abbou & Pr. A. Zahi  
Département Informatique  
FSTF  
2019-2020

# Séance 3

---

- ⊙ Fichiers
- ⊙ Récursivité

# Fichiers

---

- ⊙ Ouverture
- ⊙ Ecriture
- ⊙ Lecture
- ⊙ Fermeture

# Fichier — *Définition*

---

- ⊙ Un fichier est une entité qui permet de conserver l'information de manière **permanente**.
- ⊙ Les supports destinés pour le stockage d'un fichier sont:
  - disque dur,
  - CD,
  - flash disque (USB),
  - etc.,
- ⊙ Un fichier conserve une collection d'informations homogènes et les structures comme une suite:
  - d'**octets** — *fichiers binaires*
  - de **caractères**. — *fichiers textes*

# Fichier — *Représentation physique*

---

- ⊙ Un fichier est identifié par une chaîne de caractères:
  - ▶ Qui désigne le *nom physique* du fichier sur un support de stockage.
  - ▶ Qui est attribuée par l'utilisateur.
- ⊙ Deux fichiers sont définis par défaut pour tous les programmes C:
  - ▶ le fichier ***stdin***, qui est associé avec l'entrée standard (le clavier).
  - ▶ le fichier ***stdout*** qui est associé avec la sortie standard (l'écran).

# Fichier — *Exemples*

- L'explorateur Windows affiche 4 fichiers caractérisés par leurs noms et leurs extensions.

The screenshot shows a Windows Explorer window with the address bar set to 'Data (D:) > Enseignements > 2014-2015 > ASD'. The file list contains four items:

- exercice1.exe** (Type: Application) and **imagebot.jpeg** (Type: Image) are grouped in a red box labeled 'Fichiers binaires'. A red arrow points from this box to a text box stating: 'Fichiers considérés comme une suite d'octets'.
- main** (Type: C Source File) and **temperature** (Type: Text Document) are grouped in a green box labeled 'Fichiers Textes'. A green arrow points from this box to a text box stating: 'Fichiers considérés comme une suite de caractères'.

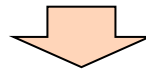
# Fichiers en C — *Principe*

---

- ⊙ Les programmes sont situés dans la Mémoire Centrale (MC).
- ⊙ Les fichiers physiques sont situés dans un support de stockage (SS).
- ⊙ La MC et les SS n'ont pas la même vitesse de lecture/écriture.

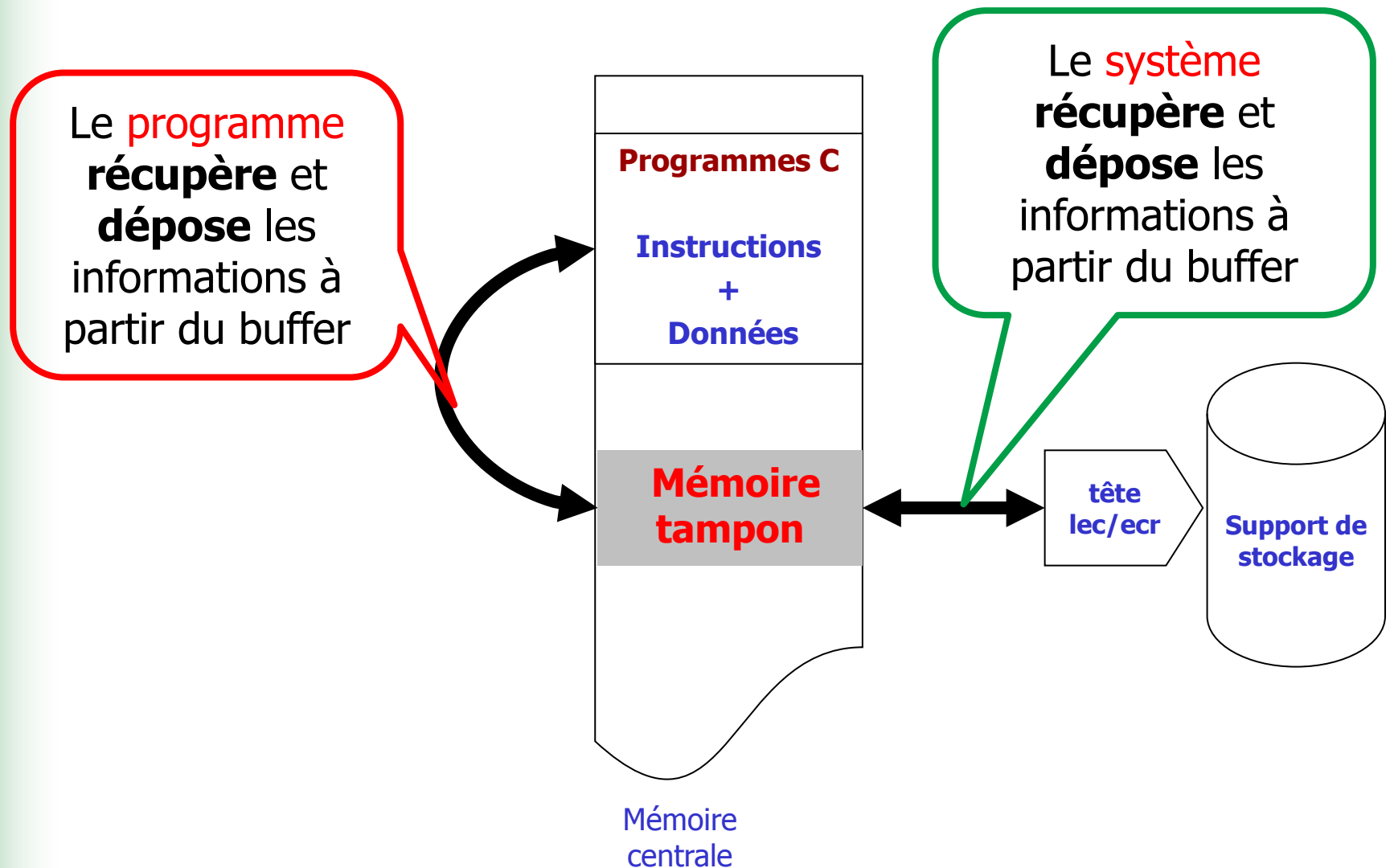


Nécessité d'utiliser un intermédiaire  
pour réduire le temps d'accès à un  
support de stockage.



Mémoire tampon  
(buffer)

# Fichiers en C — *Principe*





# Fichiers en C — *Descripteur de fichier*

---

⊙ C'est un **pointeur** de type **FILE**

▶ **FILE** — Structure prédéfinie qui contient :

- L'adresse de la mémoire tampon,
- La position courante de la tête de lecture/écriture,
- Le type d'accès au fichier,
- L'état d'erreur,
- etc.,

⊙ Assure la liaison entre le fichier physique et le programme.

# Fichiers en C — *Scénario d'utilisation*

---

- ⊙ Déclarer un descripteur de fichier;  
`FILE * fp ;`
- ⊙ Établir la connexion entre le descripteur **fp** et le nom physique du fichier.
  - ➡ la fonction **fopen** ;
    - ▶ **fp** devient le **représentant** du fichier physique dans le programme.
- ⊙ Lire/écrire à partir/dans le fichier.
  - ➡ les fonctions **fscanf/fprintf**;
- ⊙ Fermer le fichier.
  - ➡ la fonction **fclose** ;

# Fichiers en C — *fopen*

- ⊙ Ouverture d'un fichier et liaison avec le fichier physique.

- ▶ Syntaxe

## Exemple

```
FILE * fp;  
fp= fopen(non_physique, mode_ouverture) ;
```

- **fp** : Descripteur de fichier
  - **non\_physique** : Une chaîne de caractères qui contient le nom du fichier physique.
  - **mode\_ouverture** : Pointeur sur une chaîne de caractères qui indique le mode d'ouverture du fichier
- ▶ La fonction **fopen** retourne le descripteur de fichier si l'ouverture a réussi et **NULL** dans le cas échéant.

# Fichiers en C — *fopen*

- ⊙ Le **mode d'ouverture** peut être l'une des chaînes suivantes :

## Exemple

Mode	Position	Signification
"r"	début	fichier texte en lecture
"w"	début	fichier texte en écriture
"a"	fin	fichier texte en écriture
"rb"	début	fichier binaire en lecture
"wb"	début	fichier binaire en écriture
"ab"	fin	fichier texte en écriture
"r+"	début	fichier texte en lecture/ écriture
"w+"	début	fichier texte en lecture/ écriture
"a+"	fin	fichier texte en lecture/écriture
"r+b" ou "rb+"	début	fichier binaire en lecture/écriture
"w+b" ou "wb+"	début	fichier binaire en lecture/écriture
"a+b" ou "ab+"	fin	fichier binaire en lecture/ écriture

# Fichiers en C — *fclose*

---

- ⊙ Fermeture d'un fichier et rupture du lien entre le descripteur et le fichier physique.

- ▶ Syntaxe

```
FILE * fp;
```

```
.
```

```
.
```

```
fclose(fp);
```

- ▶ La fonction **fclose** retourne la valeur 0 si le fichier a été fermé, et retourne la valeur EOF s'il y a eu une erreur.
- ▶ La fonction **fclose** doit être appelée dès qu'on termine l'exploitation du fichier.

- ▶

# Fichiers en C — *fscanf*

---

## ⊙ Lecture formatée avec conversion de type.

### ▶ Syntaxe

```
FILE* fp;
```

```
·  
·
```

```
fscanf (fp, format, &var_1, &var_2, ...) ;
```

- **fp** : descripteur du fichier à partir duquel se fait la lecture.
- **format** : chaîne de caractère qui contient des caractères constants et des spécificateurs de formats.
- **var\_i** : une variable dans laquelle on récupère la valeur à lire.

# Fichiers en C — *fprintf*

---

## ⊙ Ecriture formatée avec conversion de type

### ▶ Syntaxe

```
FILE* fp;
```

```
.  
.
```

```
fprintf(fp, format, var_1, var_2, ...) ;
```

- **f p**: descripteur du fichier sur lequel se fait l'écriture.
- **format** : chaîne de caractère qui contient des caractères constants et des spécificateurs de formats.
- **var\_i** : une variable à écrire sur le fichier.

# Fichiers en C — *feof*

---

- ⊙ Lors de la fermeture d'un fichier ouvert en écriture, la fin du fichier est marquée par le symbole EOF.
- ⊙ Lors de la lecture d'un fichier, la fonction *feof* permet de détecter la fin du fichier.
- ⊙ Une valeur non nulle est retournée si l'on est à la fin du fichier, sinon, une valeur nulle est renvoyée.

## ► Syntaxe

```
FILE* fp;  
.  
.  
feof(fp) ;
```



# Fichiers en C — *Exemple 1*

---

- ⊙ Un programme qui permet de :
  - ▶ Créer le fichier « **temperature.dat** »;
  - ▶ Stocker les températures de N villes (le nombre de ville N à saisir au clavier).

# Fichiers en C — *Exemple 1*

```
/* déclaration des variables */

FILE *fp;    /*descripteur de fichier*/
int i,t;
int N;        /*nombre de villes*/

/* saisir le nombre de villes */

printf("donner le nombre de ville:");
scanf("%d",&N);

/* ouvrir le fichier en mode écriture */
fp=fopen("temperature.dat","w");

/* remplir le fichier */

if(fp!=NULL){
    for(i=0;i<N;i++){
        printf("entrer une temperature:");
        scanf("%d",&t);
        fprintf(pf,"%d\n",t);
    }
}

/* fermer le fichier */
fclose(fp);
```

# Fichiers en C — *Exemple 2*

---

- ⊙ Un programme qui :
  - ▶ Ouvre un fichier « **temperature.dat** » qui contient la température de N villes (N à saisir au clavier);
  - ▶ Charge les températures dans un tableau ;
  - ▶ Diminue les températures de 2 degré ;
  - ▶ Enregistre les données du tableau dans le même fichier.

# Fichiers en C — *Exemple*

```
/* déclaration des variables */
FILE *fp; /*descripteur de fichier*/
int T[20];
int i,cpt;

/* ouvrir le fichier en mode lecture */
fp=fopen("temperature.dat","r");

/* charger le contenu du fichier dans le tableau T*/
if(fp!=NULL){
    cpt=0;
    fscanf(fp,"%d",&t);
    while (!feof(fp)){
        T[cpt]=t;
        cpt++;
        fscanf(fp,"%d",&t);
    }
}
/* fermer le fichier */
fclose(fp);
```

# Fichiers en C — *Exemple 2*

---

```
/* modification du tableau */
    for (i=0; i<cpt; i++) {
        T[i] -=2;
    }
/* ouvrir le fichier en mode écriture*/

    fp=fopen("temperature.dat","w");

/* charger le tableau dans le fichier */

    if (fp != NULL){
        for (i=0; i<cpt; i++)
            fprintf(fp,"%d\n",T[i]);
    }

/* fermer le fichier */
    fclose(fp);
```

# Récurtivité

---

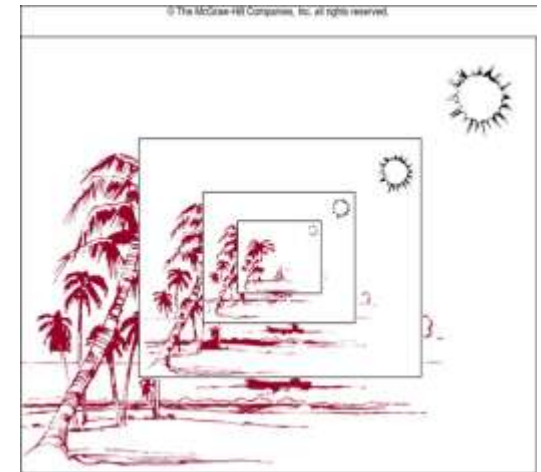
- ⊙ Illustrations
- ⊙ Problèmes rékursifs
- ⊙ Fonctions rékursives

# Récurtivité — *Illustrations*

- ⊙ La récursivité est un *principe de pensée* qui n'est pas propre à l'informatique.



l'art en fait usage pour créer des œuvres d'art .



# Récurtivité — *Illustrations*

- ⊙ La récursivité est un *principe de pensée* qui n'est pas propre à l'informatique.



On la trouve dans les motifs des végétaux



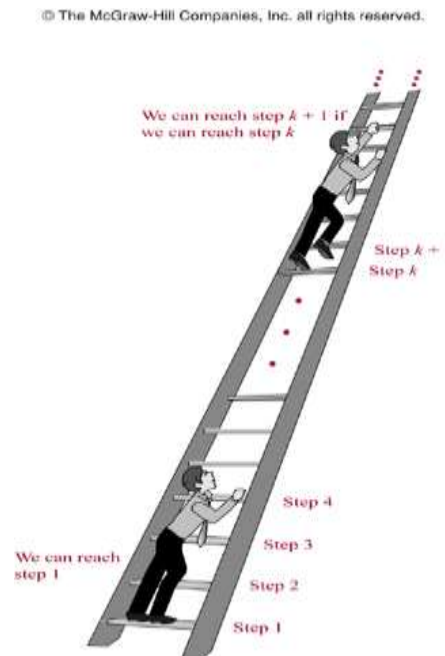


# Récurtivité — *Illustrations*

- ⊙ La récursivité est un *principe de pensée* qui n'est pas propre à l'informatique.



Pratiquée dans la vie courante pour résoudre les situations rencontrées.





# La récursivité en informatique

---

- ⊙ Technique puissante et naturelle pour :
  - ▶ Définir et décrire les *problèmes* à résoudre.
  - ▶ Mettre en œuvre les *fonctions* qui implémentent les solutions de ces problèmes.
  - ▶ Définir les *structures* qui représentent les données qui sont de nature récursives.

# Problèmes récurrents

---

- ⊙ La définition récursive d'un problème
  - ▶ trouve son formalisme dans le raisonnement par récurrence.
  - ▶ met en jeu deux parties:
    - des **cas de base**  
 définissent des sous problèmes triviaux.
    - une **équation de récurrence**  
 exprime le problème en fonction des sous problèmes avec une difficulté inférieure.

# Problèmes récurrents

## ⊙ Exemple 1 — *les suites récurrentes*

- ▶ le problème **P(n)** consiste à calculer l'expression suivante:

$$P(n) = \prod_{i=1}^n i$$

- ▶ la définition récursive de P(n) est :
  - Un cas de base :

$$P(0) = 1$$

- Une équation de récurrence :

$$P(n) = \prod_{i=1}^n i = p(n-1) * n$$

# Problèmes récurrents

## ⊙ Exemple 2 — *les suites récurrentes*

- ▶ le problème **P(n)** consiste à calculer l'expression suivante:

$$S(n) = \sum_{i=0}^n i$$

- ▶ la définition récursive de P(n) est :
  - Un cas de base :

$$S(0) = 0,$$

- Une équation de récurrence :

$$S(n) = \sum_{i=0}^n i = \sum_{i=0}^{n-1} i + n = S(n-1) + n$$

# Problèmes récurrents

## ⊙ Exemple 3 — *les suites récurrentes*

- ▶ le problème **P(n)** consiste à calculer le nombre de Fibonacci par la suite définie par:

$$\begin{cases} U(n) = U(n-1) + U(n-2) & n \geq 2 \\ U(0) = 1 \\ U(1) = 1 \end{cases}$$

- ▶ la définition récursive de P(n) est :

- Deux cas de base :

$$U(0) = 1, U(1) = 1$$

- Une équation de récurrence :

$$U(n) = U(n-1) + U(n-2)$$

# Problèmes récurrents

---

- ⊙ Exemple 4 — *le reste de la division entière*
  - ▶ le problème **P(n,d)** consiste à calculer le reste de **n** par **d** en utilisant les soustractions successives.
  - ▶ la définition récursive de  $P(n, d)$  est :
    - Un cas de base : le reste de  $n$  par  $d$  est  $n$  si  $n < d$ .
    - Une équation de récurrence : le **reste** de **n** par **d** est **exactement** le **reste** de **n-d** par **d**.

# Problèmes récurrents

## ⊙ Exemple 5 — *l'affichage à l'envers d'une chaîne de caractères*

- ▶ le problème **P(n)** consiste à afficher une chaîne de caractères à l'envers, la chaîne est saisie caractère par caractère jusqu'à la saisie du caractère '.'.
- ▶ la définition récursive de P(n) est :
  - Un cas de base : la saisie du '.' ne rien faire
  - Une équation de récurrence : saisir un caractère, **afficher** la chaîne, ensuite afficher le caractère saisi.



# Fonctions récursives

---

- ⊙ Une fonction **récursive** est une fonction qui:
  - ▶ s'appelle elle-même d'une manière directe ou indirecte.
  - ▶ Implémente une définition récursive de la manière suivante :
    - Un **test d'arrêt** qui exprime les cas de base.
    - Des **appels récursifs** qui expriment l'équation de récurrence.

# Fonctions récursives

## ⊙ Syntaxe de la 1ere forme

```
type_retour fonct_réursive(parametres)
{
    <pré_taitement>

    if (test_arrêt) resoudre_cas_de_base
    else
        fonct_réursive(parametres)

    <post_traitement>
}
```

# Fonctions récursives

## ⊙ Syntaxe de la 2eme forme

```
type_retour fonct_réursive(parametres)
{
    <pré_taitement>

    if ( ! test_arrêt )
        fonct_réursive(parametres)

    <post_traitement>
}
```

# Fonctions récursives

---

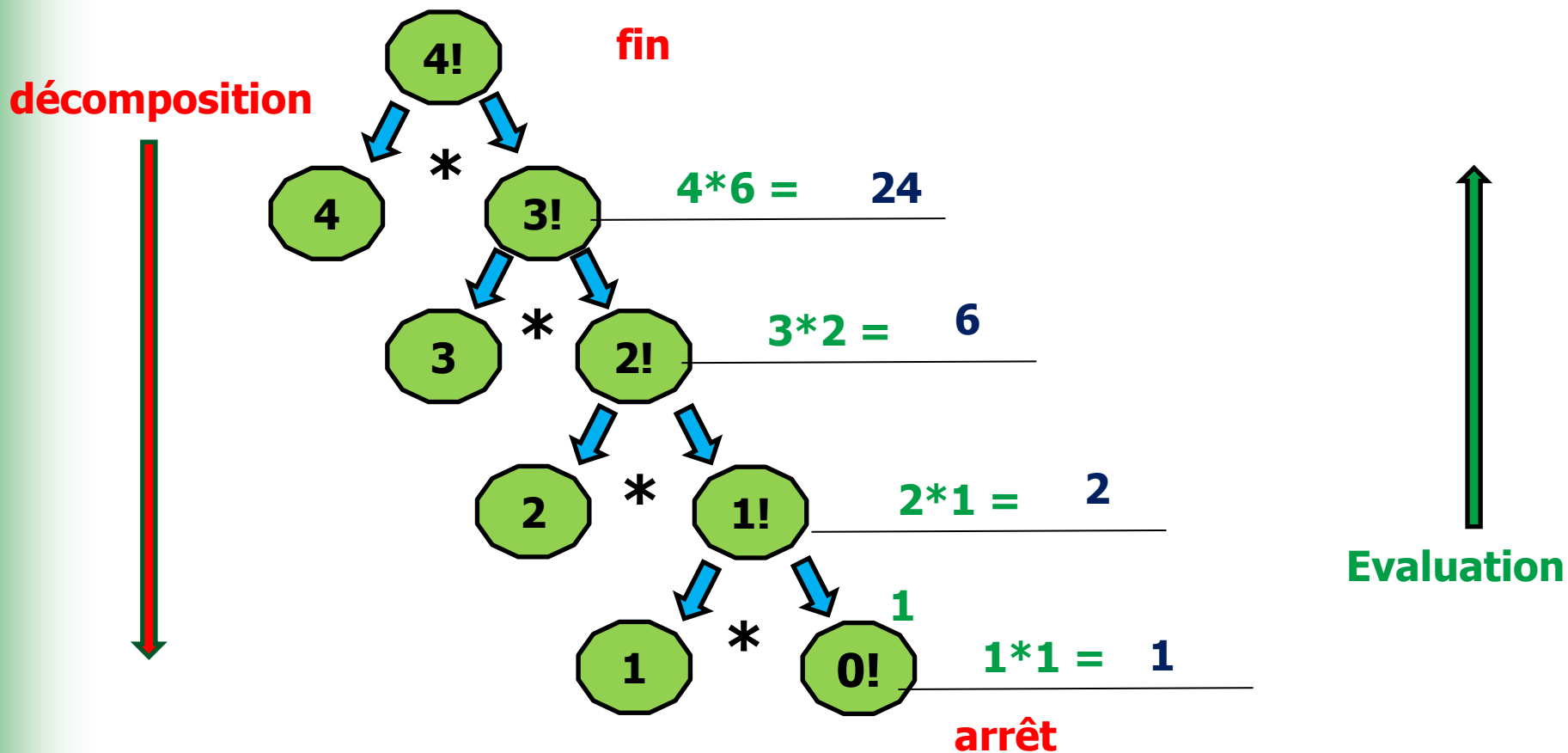
- ⊙ Exemple 1 : calcul de la factorielle d'un entier

.

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

# Fonctions récursives

## Exemple 1 : Exécution avec 4!



# Fonctions récursives

## ⊙ Exemple 2 — *calcul de la somme*

```
float somme(int n)
{
    /* aucun prétraitement*/

    if (n == 0) return 0; /* les cas de base*/

    return n+somme(n-1); /* appel récursif*/

    /* aucun post-traitement*/
}
```

# Fonctions récursives

## ⊙ Exemple 3 — *le reste de la division entière*

```
int reste(int n, int d)
{
    /* aucun prétraitement*/

    if (n < d) return n; /* les cas de base*/

    return reste(n-d,d); /* appel récursif*/

    /* aucun post-traitement*/
}
```

# Fonctions récursives

## ⊙ Exemple 4— *calcul du nombre de Fibonacci*

```
float fibonacci(int n)
{
    /* aucun prétraitement*/
    /* les cas de base*/
    if (n == 0) return 1;
    if (n == 1) return 1;

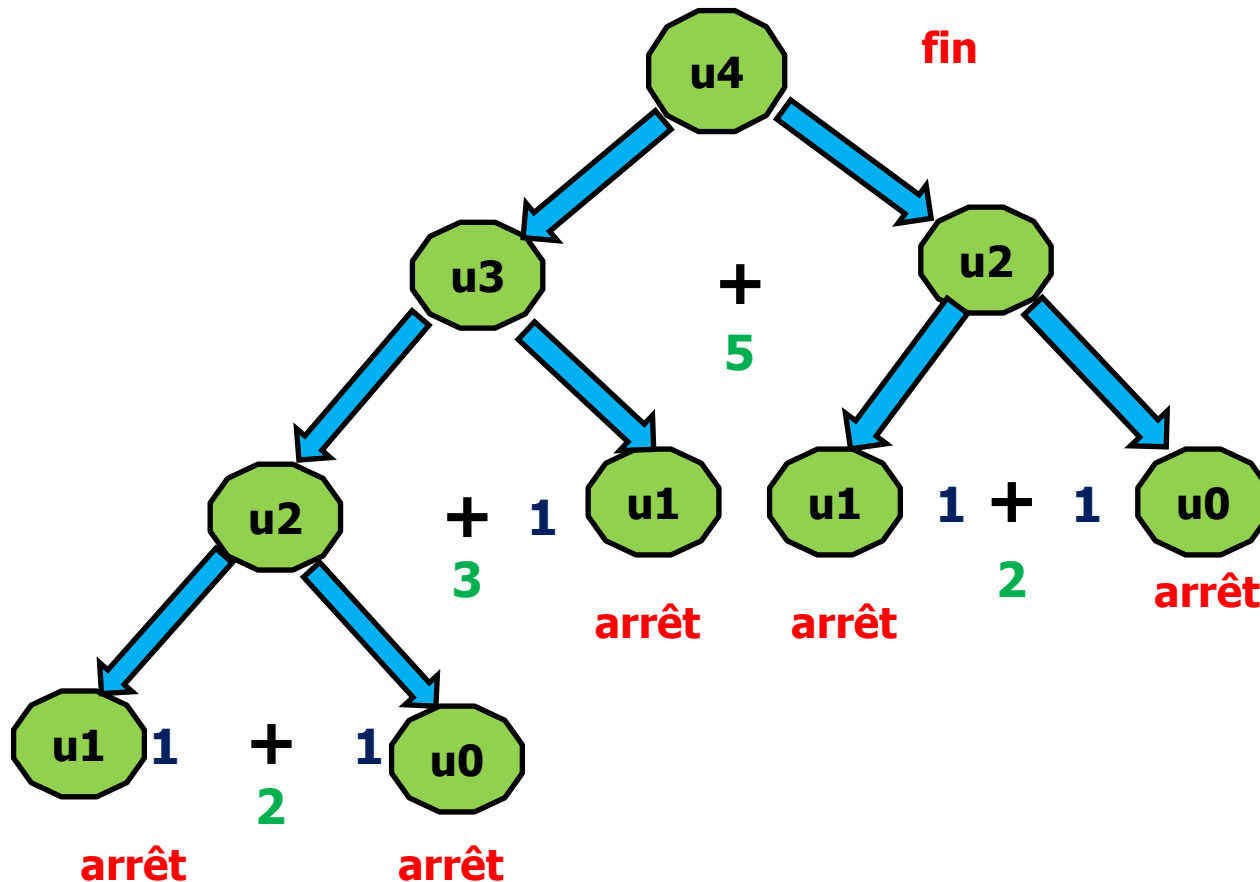
    /* appel récursif*/
    return fibonacci(n-1)+ fibonacci(n-2); ;

    /* aucun post-traitement*/
}
```



# Fonctions récursives

## Exemple 4 : Exécution avec U(4)



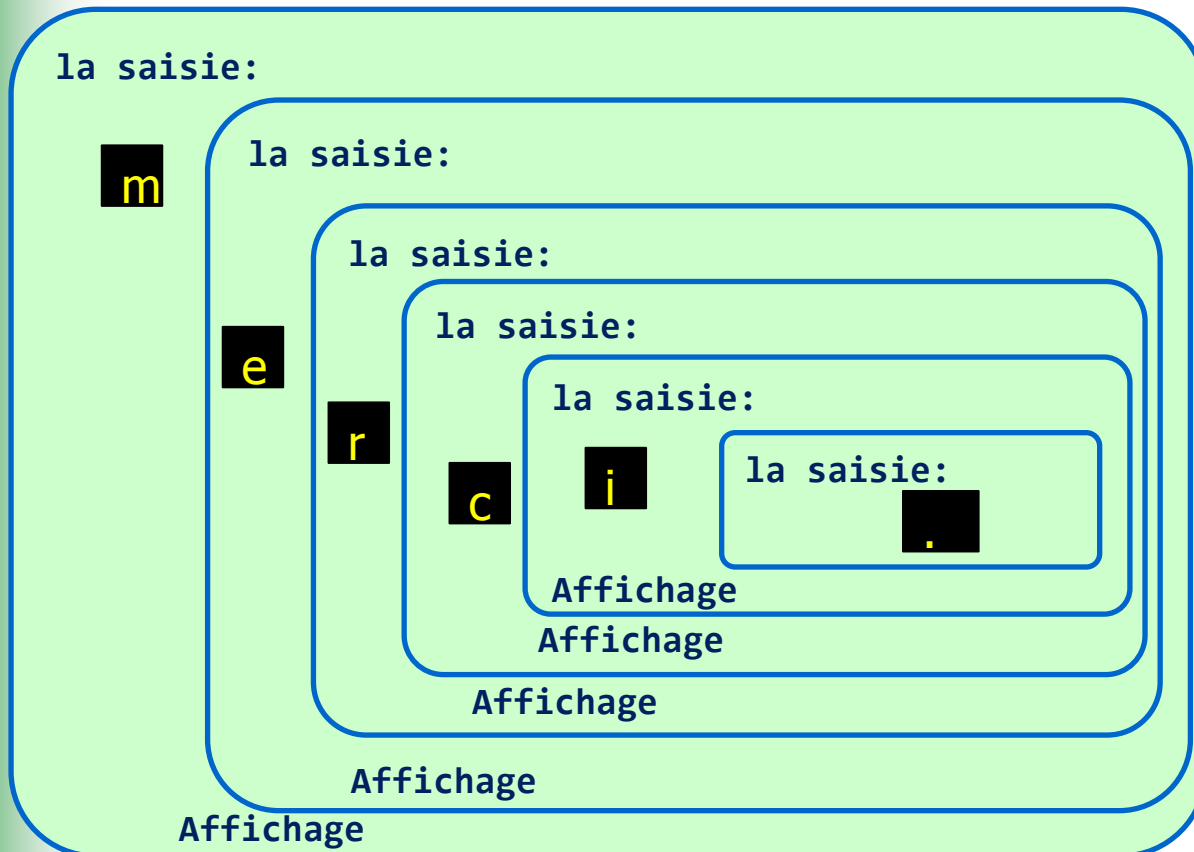
# Fonctions récursives

- ⊙ Exemple 5 — *l'affichage d'une chaîne à l'envers*

```
void aff_envers()  
{  
    char ch1;  
    /* prétraitement*/  
    ch1=getchar();  
    if (ch1!= '.'){  
        aff_envers(); /* appel récursif*/  
        printf("%c",ch1);  
    }  
}
```

# Fonctions récursives

## Exemple 5 — *l'affichage d'une chaîne à l'envers (merci)*



# Récurtivité — *Exercice*

## ⊙ *Recherche dichotomique*

- ▶ le problème **P(T, d, f, valeur)** consiste à diviser le tableau en deux parties et chercher la valeur dans les sous tableaux.
- ▶ la définition récursive de P(n) est :
  - Deux cas de base :
    - la valeur n'existe pas  $d > f$
    - La valeur se trouve dans le milieu si  $T[\text{milieu}] == \text{valeur}$
  - Équation de récurrence :

$$P(T, d, f, \text{valeur}) \rightarrow \begin{matrix} P(T, \text{milieu} + 1, f, \text{valeur}) \\ P(T, d, \text{milieu} - 1, \text{valeur}) \end{matrix}$$

# Récurtivité — *Exercice*

## ⊙ Recherche dichotomique

```
int rechercheDich(int *T, int d, int f, int val)
{
    /* prétraitement*/

    int m=(d+f)/2;
    /* les cas de base*/
    if (d >f) return -1;
    if (T[ m]== val) return m;
    /* le cas general*/
    if (T[ m]< val)
        return(rechercheDich(T,d,m-1,val);
    else
        return(rechercheDich(T,m+1,f,val);
}
```