La récursivité

1) Domaines d'utilisation :

La récursivité est une notion provenant des mathématiques.

Pour la programmation, on a besoin de connaître la récursivité pour :

. comprendre le tri rapide (Quick Sort) ;

. comprendre l'arbre binaire dont 90 % du traitement est récursif.

Il est rare qu'un programmeur doive écrire lui-même une fonction récursive. Cependant, il est profitable d'être capable d'écrire des fonctions récursives simples et de les simuler pour comprendre le principe.

2) Définition :

En pratique, une fonction est récursive si elle s'appelle elle-même.

exemple :

```
int  factoriel ( int n )
  {
      if ( n <= 1 )
              return 1 ;
      else
              return n * factoriel(n-1) ;
  }
```

3) Conseils d'écriture d'une fonction récursive :

Ces conseils sont illustrés par l'exemple suivant :

Écrire une fonction récursive permettant de calculer la somme des chiffres d'un entier n positif (exemple : n = 528, la somme des chiffres de n est 15).

Professeur M^r^ Kamel Bel Asri

1. Observer le problème afin de :

a) décrire la condition d'arrêt : quand peut-on trouver "facilement" la solution ? (une solution triviale) :

Si on a le choix entre n = 528 et n = 8, il est certain qu'on choisit n = 8. La somme des chiffres de 8 est 8.

Conclusion : Si n a un seul chiffre, on arrête. La somme des chiffres est n lui-même.

```
if ( n < 10 ) return n;
```

b) de réduire le problème à un problème d'ordre inférieur pour que la condition d'arrêt soit atteinte un moment donné :

```
somChif(528) <===>   8 + somChif(52)
             <===>   8 + ( 2 + somChif(5) )
             <===>   8 +   2 +       5
```

D'un problème de 3 chiffres, on réduit d'abord à un problème de deux chiffres : somChif(52). Celui-ci est réduit à celui d'un seul chiffre : somChif(5). On rencontre la condition d'arrêt.

```
else
  return n % 10 + somChif ( n / 10 ) ;
```

2. Écriture de la fonction :

```
int somChif ( int n )
{
   if ( n < 10 ) /* condition d'arrêt */
            return n;
   else
            /* réduction du problème : */
            return n % 10 + somChif ( n / 10 ) ;
}
```

Exercice :

Illustrer les conseils précédents pour écrire une fonction récursive qui permet de calculer le produit de deux entiers positifs a et b.

Notes :

1. Si vous avez le choix entre : 12 x 456 et 12 x 0

   lequel des deux calculs est le plus simple ?

2. 12 x 9 = 12 + 12 + 12 + ... + 12 (9 fois)
   = 12 + (12 + 12 + ... + 12) (8 fois)

   = 12 + 12 x 8

   etc ...

4) Simulation :

Ne pas utiliser les "noms" des arguments. Il est préférable d'utiliser les rangs de ces arguments (le premier paramètre, le deuxième paramètre, etc ...)

Exemple 1 :

Soit la fonction récursive suivante :

```
int facto ( int n )
{
   if ( n <= 1 ) return 1 ;
   else
         return n * facto(n-1) ;
}
```

Quel est la valeur de facto (3) ?

Exemple 1 :

Le paramètre (ici 3) est-il <= 1 ?

Non. On retourne :

```
3 * facto (2)
    -------------------------------------
    Le paramètre (ici 2) est-il <= 1 ?
    Non. On retourne  :  2 * facto(1)
                             ---------------------
                             Le paramètre (ici 1)
                             est-il <= 1 ?
                             Oui. On retourne 1
```

Ainsi facto(2) vaut 2 * 1 = 2
et facto(3) = 3 * facto(2) vaut 6.

En pratique, on fait :

```
facto(5)  <====>    5 * facto(4)
                        -------------
                        4 * facto (3)
                            ------------
                            3 * facto(2)
                                ---------
                                2 * facto(1)
                                    --------
                                        1
```

En remontant, facto(5) vaut 1 * 2 * 3 * 4 * 5 qui est 120

Remarque:

Il est préférable, si cela est plus facile, d'écrire un processus itératif (avec boucle) équivalent plutôt qu'une version récursive.

5) Exemples et exercices :

Exemple 1:

Écrire un programme permettant de saisir un entier positif et d'afficher la somme des chiffres du nombre lu. Le calcul de cette somme se fait de 2 manières différentes :

1. avec 1 fonction récursive
2. avec une fonction "itérative"

Solution :

```
#include <stdio.h>


/* fonction récursive */
int somChif ( int n )
  {
     if ( n < 10 )
           return n     ;
     else
           return (n % 10) + somChif (n / 10) ;
  }
```

```
/* processus itératif (avec boucle, ici while) */
int Somme2 (int n )
  {  int somChiffres = 0 ;

     while ( n )
        { somChiffres += n % 10 ;
          n /= 10;
        }
     return somChiffres;
   }

void main()
   {  int valeur  ;
      printf("Entrez un entier positif ");

      scanf("%d", &valeur);

      printf("\n %d (avec la récursivité)\n", somChif(valeur));
      printf("\n %d (sans récursivité)\n", somme2(valeur));
   }
```

Exercice 1 : Suite Fibonacci.

La suite Fibonacci est définie comme suit :

```
f(1)  = 1
f(2)  = 1

f(3)  = 2
f(4)  = 3
f(5)  = 5
f(6)  = 8
f(7)  = 13
etc ...
```

Mathématiquement :

```
          |  1    pour n = 1 et n = 2 (convention)
f(n)  =   |
          |  f(n-1) + f(n-2)  pour n =3, 4, 5, 6, ...
          |  (somme des deux derniers termes)
```

Écrire un programme en C permettant de :

- saisir un entier n >= 1
- de calculer et d'afficher f(n) en utilisant :

  a) la récursivité
  b) le processus itératif

Avec n = 30, quel est le processus le plus rapide ?

Exemple 2 :

Déterminer du k ième chiffre à partir de la droite d'un entier n > 0 :

```
  Le 3ième chiffre à partir de la droite de 8724   est 7
  Le 5ième chiffre à partir de la droite de 21327  est 2
```

Écrire une fonction récursive chiffre(n, k) qui permet de retourner le k-ième chiffre à partir de la droite de n.

Observations :

1. Si k = 1, la solution est triviale. Le chiffre recherché est le dernier chiffre de n, c'est-à-dire n % 10 :

```
      if ( k == 1 ) return n % 10 ;
```

   C'est la condition d'arrêt.

2. Si k > 1, exemple k = 3, on a :

```
      chiffre (8724, 3)
<==>  chiffre ( 872, 2)   (872 est 8724 / 10, 2 est 3-1)
<==>  chiffre (  87, 1)   (87  est 872  / 10, 1 est 2-1)
```

Solution :

```
  int chiffre ( int n, int k )

  {
      if ( k == 1 ) /* le premier chiffre à partir de la droite */
           return n % 10 ;
      else
           return chiffre ( n / 10, k - 1 ) ;
  }
```

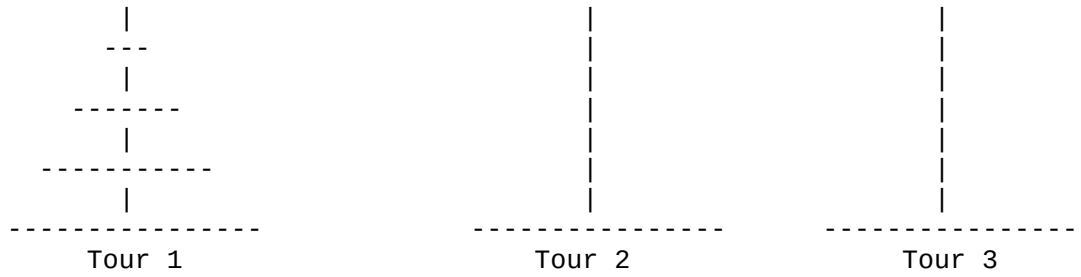
Exemple 3 : Tours de Hanoi.

Supposons qu'on dispose de 3 disques à la tour 1 et on aimerait les déplacer à la deuxième tour en utilisant la tour 3 comme intermédiaire.

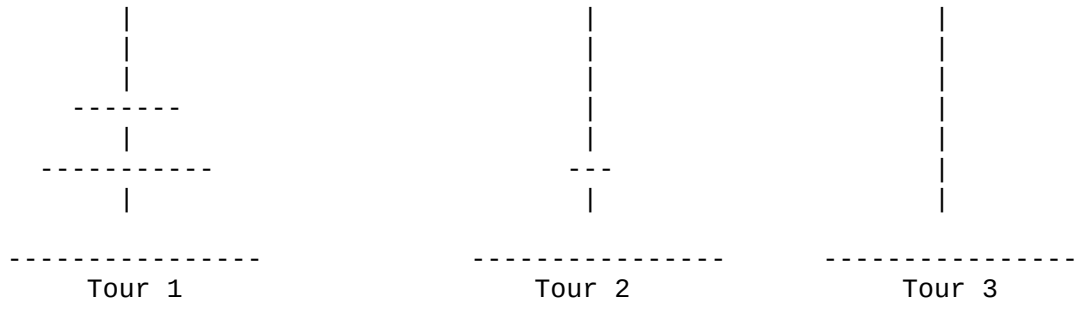
Les règles du jeu sont les suivantes :

- les disques ont un diamètre différent
- on déplace un disque à la fois
- on n'a pas le droit de placer un disque de diamètre supérieur sur un de diamètre inférieur.

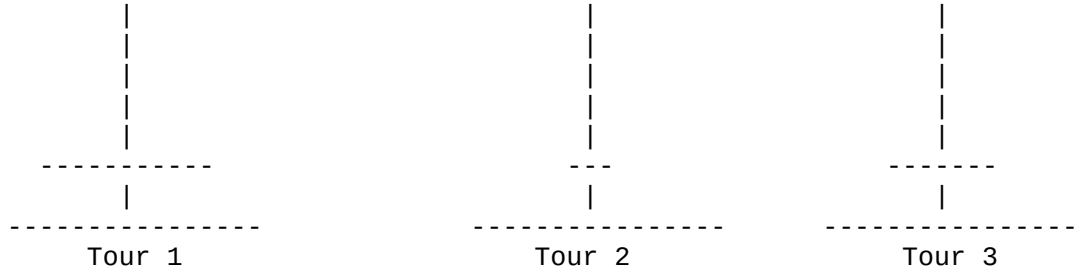
état 0 :



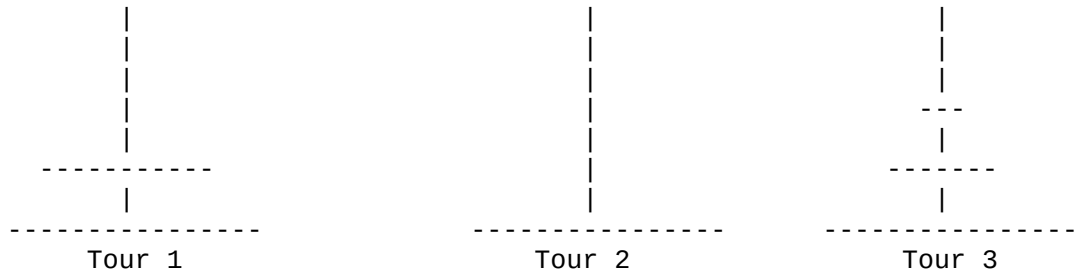
état 1 : déplacer un disque de Tour 1 à Tour 2



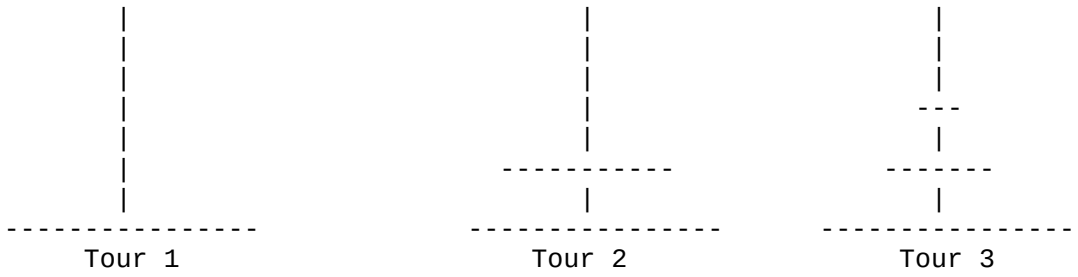
état 2 : déplacer un disque de Tour 1 à Tour 3



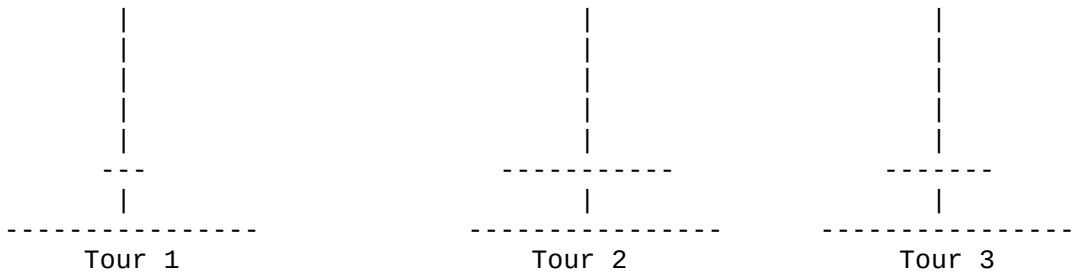
état 3 : déplacer un disque de Tour 2 à Tour 3



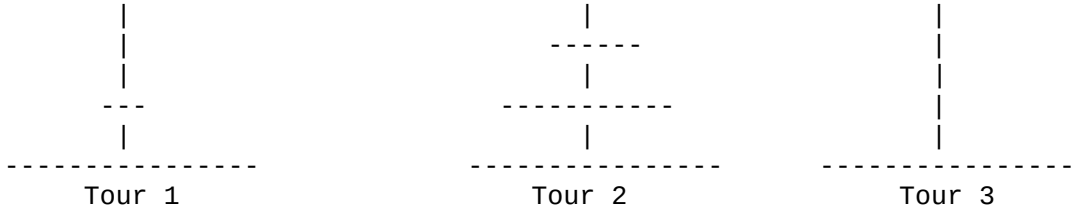
état 4 : déplacer un disque de Tour 1 à Tour 2



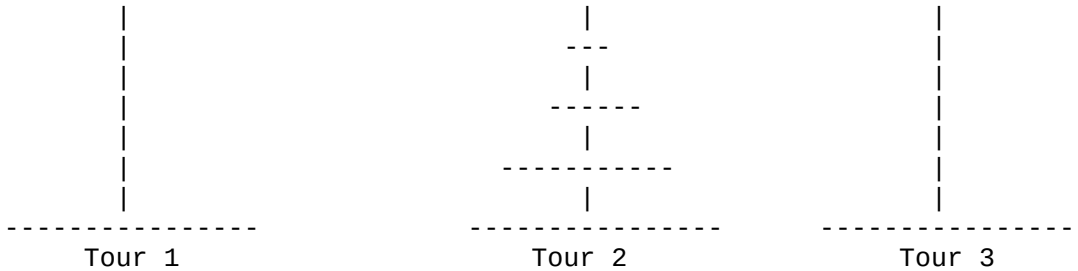
état 5 : déplacer un disque de Tour 3 à Tour 1



état 6 : déplacer un disque de Tour 3 à Tour 2



état 7 : déplacer un disque de Tour 1 à Tour 2



Le processus sera plus compliqué et mêlant quand le nombre de disques deviendra grand.

Professeur M^r^ Kamel Bel Asri

Solution :

```
#include <stdio.h>

   void deplacer (int n, int t1, int t2, int t3)
     {
        if ( n == 1 )
               printf("De la tour %d à la tour %d\n", t1, t2) ;
        else
            { deplacer (n-1, t1, t3, t2) ;
              deplacer ( 1 , t1, t2, t3) ;
              deplacer (n-1, t3, t2, t1) ;
            }
     }


void main()

{  int nbDisques ;

   printf("Entrez le nombre de disques à déplacer ");
   scanf ("%d", &nbDisques);

   deplacer ( nbDisques, 1, 2, 3);

 }
```

Note :

Une autre version possible de deplacer est :

```
   void deplacer (int n, int t1, int t2, int t3)
     {
        if ( n > 0 )
          {
             deplacer (n-1, t1, t3, t2) ;
             printf("De la tour %d à la tour %d\n", t1, t2) ;
             deplacer (n-1, t3, t2, t1) ;
          }
     }
```

exécution :

```
Entrez le nombre de disques à déplacer 3

De la tour 1 à la tour 2
De la tour 1 à la tour 3
De la tour 2 à la tour 3
De la tour 1 à la tour 2
De la tour 3 à la tour 1
De la tour 3 à la tour 2
De la tour 1 à la tour 2
```

Exercice:

Simuler l'appel suivant :

```
deplacer (3, 1, 2, 3) ;
```

6) Tri rapide (Quick Sort) :

Supposons qu'on dispose du tableau t qui contient les 9 éléments suivants à trier :

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]	t[7]	t[8]
43	75	23	46	55	12	64	77	33

L'idée de la méthode QuickSort est simple :

1) Pour la première fois, on tri le tableau au complet, c'est-à-dire de l'indice 0 à l'indice (nbElement - 1) (ici 8). Ceci correspond à l'appel suivant :

```
quickSort(t, 0, nbElement - 1) ;
```

2) On partitionne le tableau en deux côtés d'une valeur du pivot.
A gauche, on trouve des valeurs <= la valeur du pivot.
A droite, on trouve les valeurs > la valeur du pivot.

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]	t[7]	t[8]
12	33	23	43	55	46	64	77	75

```
valeurs <= 43          ^          valeurs > 43
                       |
                       |
                valeur du pivot
          ( 43 trouve sa bonne place
            à l'indice du pivot 3   )
```

On verra plus loin la technique de la partition.

On réapplique de nouveau le QuickSort pour la partie gauche (indices 0 à 2) et la partie droite (indices 4 à 8) du tableau. Cette réapplication nous montre le concept de la récursivité.

Quelle est la condition d'arrêt ?

- on tri une partie du tableau quand elle contient au moins deux valeurs à mettre en ordre. C'est-à-dire : l'indice gauche est inférieur à l'indice droite. Sinon, la partie est triée.

Quelle est la réduction du problème à celui d'un ordre inférieur ?

- au lieu d'appliquer le QuickSort sur tout le tableau, on l'applique sur deux sous-tableaux : celui à gauche de l'indice du pivot et celui à droite de cet indice. Chacun de ces deux sous-tableaux contient moins d'éléments, ce qui fait accélérer le processus de tri (i.e. de la répartition).

Exemple1 :

Cet exemple permet de voir la programmation de la méthode du tri rapide. On utilise un petit tableau de 7 éléments dont la simulation du tri sera présentée en classe :

```
int t[10]     = { 20, 15, 80, 32, 10, 90, 46 } ;
int nbElem   = 7 ;
```

Solution :

```
/* Fichier : Q_Sort1.A95 : Tri rapide avec Quick Sort

   Matière principale : le tri rapide
*/

#include <stdio.h>

void partitionner(int t[], int, int, int *) ; /* prototype */

/* données simple pour démontrer que le tri rapide fonctionne */
int t[10]     = { 20, 15, 80, 32, 10, 90, 46 } ;
int nbElem   = 7 ;

   void afficher(int t[], int nbElem, char * quand)
   { int i ;

     printf("\nContenu du tableau %s\n\n", quand);

     for (i = 0 ; i < nbElem ; i++)
       printf(" %3d",  t[i]);
     printf("\n\n");
   }

    void quickSort (int t[], int gauche, int droite)

   {
     int  indPivot ;

     if (gauche < droite)
       { partitionner (t, gauche, droite, &indPivot);
        quickSort(t, gauche, indPivot - 1 );
        quickSort(t, indPivot+1, droite);
       }
   }
```

```
   void echanger(int * a, int * b) /* transmis par pointeur */
   {

     int tempo  ;

       tempo = *a , *a = *b, *b = tempo ; /* opérateur séquentiel "," */

   }


   void partitionner(int t[], int debut, int fin, int * indice)
   {
     int   g = debut , d = fin ;
     int   valPivot = t[debut];

     do
      { while (g <= d && t[g] <= valPivot) g++ ;
       while (t[d] > valPivot) d-- ;

       if (g < d) echanger(&t[g], &t[d]);
      }
     while (g <= d );

     echanger(&t[debut], &t[d]);
     *indice = d ;
   }

void main()
{


   afficher(t, nbElem, "avant le tri");

   /* Appel du tri rapide */
   quickSort( t, 0, nbElem - 1) ;

   afficher(t, nbElem, "après le tri");


   printf("\nAppuyez sur Entrée ");
   fflush(stdin);
   getchar();
}
```

Exécution :

```
Contenu du tableau avant le tri

  20  15  80  32  10  90  46


Contenu du tableau après le tri

  10  15  20  32  46  80  90


Appuyez sur Entrée
```

8) Exercices sur la récursivité :

Exercice 1 : Tri d'un tableau de structure.

Remplacer le tri par sélection dans l'exemple "Struct2.C" (chapitre 5) par le tri rapide.

Exercice 2

Écrire une fonction récursive pour la recherche dichotomique dans un tableau t qui contient exactement n entiers triés.