Algorithmique S4 P3

Rappels de langage C :

Les pointeurs :

une variable de type pointeur est destinée à stocker l'adresse d'une autre variable.

```
int * p ; // p est destiné à stocker l'adresse d'une variable de type int
int i ;
p = &i ; // p contient l'adresse de i
```

on peut alors accéder au contenu de i par *p

Les structures :

une structure permet de regrouper sous même nom des variables de type différents :

```
struct Enregistrement {
            int Numero ;
            char Nom[80];
            char Prenom[80];
            float moyenne ;
            int age ;
          };

    struct Enregistrement E1 , E2 ;
```

Accès au champs : **opérateur .**

```
printf("Nom : %s\n" , E1.Nom);
E1.moyenne = 12.3 ;
```

opérateur ->

Lorsqu'on dispose d'un pointeur sur une structure on accède aux champs par **->** :

```
struct Enregistrement * pe ;
 pe = & E1 ;

 pe->age = 20 ;
 printf("Prenom : %s\n", pe->Prenom);
```

Tableau de structures et pointeurs :

```
struct Enregistrement * pe ;
struct Enregistrement E[10] ;
int i ;

pe = E ;  // pe "pointe" E[0]
for ( i=0 ; i <10 ; i++ ) {
   pe->age = 20 ;
   pe++ ; // pe "pointe" l'enregistrement suivant
 }
```

Allocation dynamique :

La fonction malloc demande au système d'exploitation une zone mémoire contiguë.

Si la mémoire n'est pas disponible malloc retourne NULL , sinon on obtient l'adresse de début de la zone allouée.

La mémoire doit ensuite être libérée par un appel à la fonction free.

Exemple :

```
struct Enregistrement * pdebut ;
struct Enregistrement * pe ;

pdebut = malloc(20*sizeof(Enregistrement));

if ( pedebut == NULL ) return -1 ;

for ( pe=pdebut ; pe<pedebut+20 ; pe++ ){
  pe->moyenne = 10 ;
  strcpy(pe->Nom , "noname" );
}
…
free( pdebut ) ;
```

Récursivité :

Une fonction récursive est une fonction qui s'appelle elle même.
Il faut bien sûr prévoir une condition d'arrêt pour ne pas entrer dans une suite d'appel infini.
La programmation récursive est souvent élégante et permet d'écrire des programmes concis.
Par contre elle est souvent inefficace car les appels de fonctions sont des opérations longues.
Elle n'est jamais nécessaire car on peut toujours trouver un algorithme itératif équivalent à un algorithme récursif.
Voici pour exemple une fonction factorielle récursive :

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