Devoir surveillé [X]	Examen [ ]	Session: principale [X] de contrôle [ ]
Matière : Algorithmique		Semestre: 1
Enseignant(s) : Riadh ROBBANA		Date: Avril 2013
Filière(s) : MPI		Durée: 1h 30 mn
Barème : Ex 1 : 6 pts, Problème : 14 pts		Documents: autorisés [ ]
Nombre de pages : Deux pages		non autorisés [X]

Exercice 1: (6 pts)

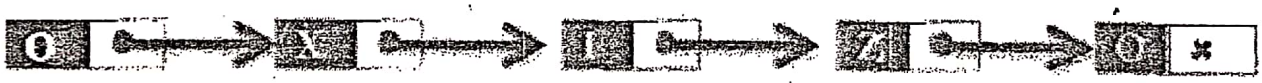
Soit la structure suivante d'une liste chaînée :

```
typedef struct liste{
        char caract ;
        struct liste * suivant ;
} Liste ;
```

Ecrire une fonction qui permet de vérifier si une liste est palindrome ou non.

Exemple :

→ Cette liste est palindrome



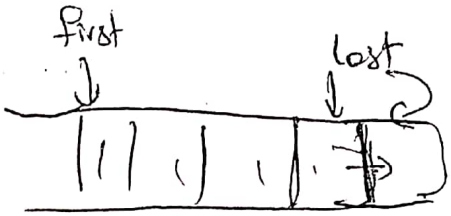
→ Cette Liste n'est pas palindrome.

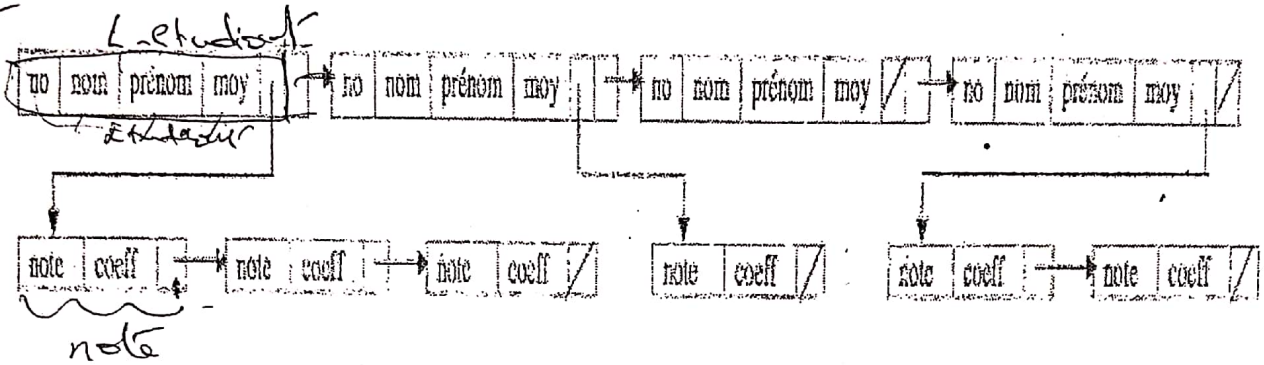
Exercice 2 : (14pts)

Le département dans lequel vous êtes inscrit souhaite gérer les notes de ses étudiants. Les étudiants ont pour identifiant leur numéro d'étudiant. Ils ont un nom et un prénom. Ces informations sont stockées dans une liste simplement chaînée dont chaque élément comporte aussi un champ « moy» pour la moyenne de l'étudiant et un champ « eval» qui est un pointeur sur sa liste de notes. La liste de notes de chaque étudiant est aussi une liste chaînée dont la tête est le champ eval de la cellule de l'étudiant. On dispose des déclarations suivantes :

```
typedef struct {
        int no ;
        char nom[50] ;
        char prenom[50] ;
        float moy ;
        notes *eval ;
}Etudiant ;
```

```
typedef struct {
        float  note ;
        float coeff ;
} Notes ;
typedef struct L_etudiant{
        Etudiant e;
        struct L_etudiant *suivant ;
}Liste ;
```





Remarques :

Cette notation équivaut à Nil

Il se peut qu'un étudiant n'ait pas encore de note. C'est le cas du 3ème étudiant de la liste de l'exemple ci-dessus. Le pointeur *eval* est égal à Nil.

a. Ecrire la fonction **Etudiant SaisirEtudiant()** qui permet de saisir un étudiant à la fois (tous les champs sont obligatoires et non vides sauf la moyenne et note).

b. Ecrire une fonction **Liste* Construire_liste_triee()** qui permet de construire la liste des étudiants triée par leurs numéros.

c. Ecrire une fonction *récursive* **Etudiant rechercher (Liste* l, int numero)** qui permet de chercher un étudiant. L'étudiant est repéré par son numéro.

d. Ecrire une fonction **Liste* Saisie_note(Liste*l)** qui permet la saisie de la notes des étudiants qui varient entre 0 et 20. L'étudiant est repéré par son numéro.

e. Ecrire la fonction *récursive* **Liste* Calcul_moy(Liste* l)** qui calcule la moyenne des étudiants.

f. Ecrire une fonction **Liste* Update_liste (Liste* l)** qui permet de mettre à jours **UN champs** donnée dans la liste. Ce champ peut être soit le coefficient, soit la moyenne, soit la note. Si le champ est égal à « coef », tous les étudiants doivent avoir le même coefficient dans la matière.

g. Ecrire une fonction **Liste* Suppression (Liste* l)** qui permet de supprimer un étudiant repéré par son numéro.

DS 2013 (algo)

Exercice 1:

```
int verif (liste *l ) { liste p=l;
        int i=0 , j=0 ;
While (j!=i) { liste p=l; while ( p -> next) { p = p-> next ;
                                                i++ ; }
    if (p -> caract == l -> caract ) { j=i ;  l = l -> next;
                                       p -> next = NULL;
                                       j-- ; }
    } if ( i == j ) return 1 ; else return (0) ; }.
```

Exercice 2:

```
etudiant saisi_etude(){
        etudiant e;
do { i=0
   printf ("nom") ; if s-canf ("%s", e.nom)  i++ ;
   printf ("prenom"),   s-canf ("%s", e.prenom)  ;
   printf ("num") ;     s-canf ("%d", &e.no)  ;
   printf ("moyenne");  s-caf ("%f", &e.moy), ; }

   While (i < 4) ;
   printf ("note"); s-caf ("%f", & e.eval.note) ;
   printf ("coef") ; s-canf ("%f", & e.eval.note) ;
   return e ; }.

liste - constitue_liste_tri() { etudiant q ; liste * p ; R ;
liste * l = (liste *) malloc (sizeof (liste)) ;   (R = l ;)
        q = saisi_etud () ;
do { do { p->e = saisi_etud () ;  if ((q.no < l->e.no)) {
                                  l->e = q ; q = saisi_etud () ; }
else { p->e = p->e ; p = p-> next ;  l = l->next } } while (p -> next) ;
        return (R) ; }
```

```
~~Etudiant - recherche (liste * l , int numero) {~~

    if (numero == l->e.no) { return (e); }
    else if (l == NULL) return (NULL);
    else return (ab recherche (l->suivant , numero)); }.

• liste * saisir_note (liste * l) {
    do { printf ("saisir note");
         scanf ("%f", & l. e.eval.note); }
    while (l->e.eval.note > 20);
    return (l); }.

• liste * calc_moy (liste l) {.
  recursif ?!
    ~~if (l == NULL) {l->e.moy = 0;~~
    ~~else~~ {    while (l != NULL) { float s = 0;   Note *T;
                 if (T == NULL) {
                     l->e.moy = 0; }
                 else { l->e.moy = l->e.note * l->e.eval.coeff ;
                        s += l->e.eval.coeff ; }.
              }

• liste * Update_liste (liste * l) {.   int x ;
    do { printf ("saisir le choix ");
         scanf ("%d", & x); }  while (x > 3 || x < 1)

    switch (x) {
                    scanf ("%f", & c);   do { l->e.eval.coef = c ;
    case 1 : { ~~l->e.eval.coeff = c;~~        l = l->suivant ; }
                                         while (l != NULL)
                                         break ; }.
    case 2 : { scanf ("%f", & M) ; l->e.moy = M ; break ; }
    case 3 : { scanf ("%f", & N) ; l->e.eval.note = N ; break
    }.
```

bibliothèque <stdlib.h>

Il est obligatoire d'initialiser une liste chaînée : NULL

pour vérifier si une liste chaînée est initialisé vide ou pas :

return (list == NULL) ? 1 : 0 ; 1 : elle est vide sinon zéro

- après la suppression d'un élément d'une liste chaînée on doit la libérée

free (liste) ;

- nbre d'éléments dans une liste : int nb_elect (LiS list) {

if (list == null) return 0;

else return (nb_elect (list -> next) + 1);

}

adr = malloc (100 * sizeof (long)) ; : allocation dynamique d'un tableau

* (adr + i) <=> adr [i]

-liste linéaire.

typedef struct noeud { int x ; pointeur sur prochain suivant :

struct noeud * suivant ; } Noeud ;

typedef Noeud * liste ; // pointeur sur Noeud

liste Pnoeud , p ;

p = ~~malloc~~ (Noeud *) malloc(sizeof (Noeud)) ;

free (p) ;

~~initialisation d'une liste vide~~ :

initialisation liste chaînée vide.

liste ma_liste = null ;

Noeud * maliste = null ;

struct noeud * maliste = null ;

l -> x ≡ *l.x.

flèche si l'on définit comme un pointeur sinon l.x