Travaux Dirigés – TD 10

Pour ce TD, nous considérons qu'un élément d'une liste chaînée simple est le suivant :

```
typedef struct element
{
    char nom[30];
    struct element* suivant;
} element;
```

et l'élément d'une liste doublement chaînée est le suivant :

```
typedef struct element_circ
{
    char nom[30];
    struct element_circ *suivant, *precedent;
} element_circ;
```

Exercice 58. Listes circulaires simples et doubles

Écrire les fonctions et les procédures suivantes pour les listes circulaires simplement chaînées.

- `void afficher (element* liste)`
- `element first_element(element* liste)`
- `int taille(element* liste)`
- `element * remove_first (element * liste )`
- `element* add_first(element* liste, char nom[])`
- `void liberer(element* liste)`
- `element* acceder(element* liste, char nom[])`
- `element* supprimer(element* liste, char nom[])`
- `void remove_last(element* liste)`

Réécrire les mêmes fonctions pour une liste circulaire doublement chaînée.

Exercice 59. Inversion des listes

Écrire la procédure `void inverser_liste(element** liste)` qui permet d'inverser une liste circulaire simplement chaînée.

Écrire la même procédure `void inverser_liste_circ(element_circ** liste)` pour une liste circulaire doublement chaînée

Exercice 60. Guide de bâtiments à l'UTT

Un visiteur arrive à l'UTT et se perd dans l'ellipse. Il demande votre aide moyennant un programme. Chaque bâtiment a un successeur et un prédécesseur selon la carte de l'UTT. Les bâtiments en dehors de l'ellipse ne sont pas considérés.

— Quelle type de liste doit-on utiliser ? Définir la structure de liste dont l'élément est `batiment`.

— Écrire la procédure `void plusCourtChemin(batiment* liste, batiment depart, batiment arrivee)` qui permet de donner la liste la plus courte des bâtiments par lesquels l'utilisateur doit passer pour atteindre le bâtiment `arrivee` à partir du bâtiment `depart`.