NIVERSITÉ de Marne-la-Vallée

Travaux Pratiques d'algorithmique n°7

Cours d'Informatique de Deuxième Année

—DEUG Sciences M3-MI—

Allocation mémoire et listes chaînées

Le but de ce TP est de gérer un petit répertoire d'étudiants grâce aux listes chaînées. Les principales opérations à réaliser seront les suivantes :

- la saisie des coordonnées d'un étudiant ;
- l'ajout d'un étudiant dans le répertoire ;
- la suppression d'un étudiant du répertoire ;
- la visualisation et le tri du répertoire.

▸ **Exercice 1.** ***(saisie des coordonnées d'un étudiant)***
Un étudiant est caractérisé par son nom, son prénom, son numéro de TD, et son numéro de TP. On définit le type `Etudiant` *approprié :*

```
struct etudiant
{
  char * nom;
  char * prenom;
  unsigned int notd;
  unsigned int notp;
};
typedef struct etudiant Etudiant;
```

Écrire les fonctions suivantes :

- *`Etudiant* AllouerSaisirEtudiant(void)` qui effectue l'allocation mémoire nécessaire à la saisie des coordonnées d'un étudiant et qui effectue cette saisie (NB : on veillera à allouer le minimum nécessaire de mémoire, pas plus qu'il n'en faut) ;*
- *`void AfficherEtudiant(Etudiant e)` qui affiche les coordonnées d'un étudiant ;*
- *`int CompararEtudiants(Etudiant e1, Etudiant e2)` qui retourne un entier négatif, nul ou positif selon que le premier étudiant est "inférieur", "égal" ou "supérieur" au second (le critère est le nom puis le prénom) ;*
- *`void LibererEtudiant(Etudiant *e)` qui libère **toute** la mémoire allouée lors de la saisie d'un étudiant.*

▸ **Exercice 2.** ***(gestion de répertoires)*** *On souhaite gérer un répertoire sous forme d'une liste chaînée de fiches d'étudiants. Le type* `Repertoire` *est défini par :*

```
struct fiche
{
  Etudiant     *etudiant;
  struct fiche *suivant;
};
typedef struct fiche Fiche;
typedef Fiche * Repertoire;
```

Écrire les fonctions suivantes :

- *`Repertoire AjouterFicheTete(Repertoire rep, Etudiant * e)` qui alloue une fiche, lui affecte l'étudiant* `e` *et l'ajoute en tête du répertoire ; cette fonction retournera le répertoire résultat.*
- *`void LibererFiche(Fiche *f)` qui libère toute la mémoire d'une fiche et de son étudiant.*
- *`Fiche* MembreRepertoireRec(Repertoire rep, Etudiant e)` qui recherche récursivement un étudiant dans un répertoire ; cette fonction retournera l'adresse de la fiche contenant l'étudiant recherché, ou* `NULL` *s'il n'est pas présent. (Faites de même itérativement dans une fonction* `MembreRepertoireItr` *de même prototype.)*
- *`Repertoire AjouterEtudiant(Repertoire rep, Etudiant *e)` qui ajoute un étudiant dans un répertoire ; cet ajout s'effectuera sans duplication : si l'etudiant est déjà présent, on veillera à mettre à jour ses numéros de TD/TP (penser alors à libérer* `e`*) ; cette fonction retournera le répertoire résultat.*
- *`Repertoire SupprimerEtudiantRec(Repertoire rep, Etudiant e)` qui supprime récursivement un étudiant d'un répertoire s'il est présent ; cette fonction retournera le répertoire résultat.*
- *`void AfficherRepertoireRec(Repertoire rep)` qui affiche récursivement le contenu d'un répertoire. (Faites de même itérativement dans une fonction* `AfficherRepertoireItr` *de même prototype.)*
- *`void LibererRepertoireRec(Repertoire rep)` qui libère récursivement toute la mémoire allouée pour un répertoire. (Faites de même itérativement dans une fonction* `LibererRepertoireItr` *de même prototype.)*

▸ **Exercice 3.** ***(tri du répertoire)*** *En vous servant d'un des algorithmes de tri que vous avez appris (par exemple, le tri par sélection), écrivez une fonction `Repertoire TrierRepertoire(Repertoire rep)` de tri du répertoire.*

Cette fonction pourra être itérative ou récursive ; elle retournera le répertoire résultat. Attention à ne pas perdre de mémoire en route !