Pratique du C :
Pointeurs : structures autoréférentes

Licence Informatique — Université Lille
Pour toutes remarques : Alexandre.Sedoglavic@univ-lille1.fr

Semestre 4 — 2016-2017

Exercice 1 — Recherche du pgcd de deux entiers positifs sans utiliser l'algorithme d'Euclide.

1. Donner une structure de donnée de type `cell_t` permettant de construire une liste chaînée permettant de stocker des entiers positifs.
2. Donner la définition de la fonction de prototype :

   ```
   cell_t liste_des_diviseurs(unsigned int);
   ```

   qui retourne la chaînée des entiers divisant le paramètre.
3. Donner la définition de la fonction de prototype :

   ```
   cell_t elements_communs(cell_t *, cell_t *);
   ```

   qui retourne une liste composé des éléments communs aux listes passées en paramètres. Ces listes étant construitent par la fonction précédente, elles sont triées. Cette fonction détruit les listes passées en paramètres.
4. Donner la définition de la fonction de prototype :

   ```
   unsigned int plus_grand_element(cell_t *) ;
   ```

   qui retourne le plus grand élément de la liste passée en paramètre et qui détruit cette dernière. Cette liste étant produite par la fonction de la question précédente, elle est aussi triée (attention à l'ordre).
5. Donner la définition de la fonction de prototype :

   ```
   unsigned int pgcd_inefficace(unsigned int,unsigned int);
   ```

   qui retourne le plus grand commun dénominateur des entiers passés en paramètre.

Exercice 2 — Matrice creuse.

La taille d'une matrice carrée creuse n'est pas connue à la compilation mais seulement lors de sa création. Les coefficients de cette matrice sont des entiers machines signés.

On représente une matrice carrée creuse sous la forme d'une liste chaînée dont chaque élément correspond à une ligne de la matrice et contient comme information l'indice de ligne et la représentation d'une ligne. Cette liste est triée suivant la valeur de l'indice de ligne.

Une ligne est représentée par une liste chaînée stockant les éléments non nuls de cette ligne. Un élément d'une ligne contient comme information le couple (indice de colonne, coefficient). Chaque liste est triée suivant la valeur de l'indice de colonne.

Questions.

1. Donnez la déclaration des types `ligne_t` et `element_t` correspondant à une ligne et à un élément de la liste chaînée représentant une ligne.
2. Donnez la déclaration du type `MatriceCreuse_t` représentant une matrice carrée creuse comme décrit ci-dessus.
3. Donnez la définition d'une fonction de prototype `void freemat(MatriceCreuse_t *)` qui libère l'espace mémoire associé à la matrice creuse passée en paramètre.
4. Donnez la définition d'une fonction `void multscal(int a, MatriceCreuse_t M)` qui réalise la multiplication de `M` par le scalaire `a`. Les éléments de `M` sont directement modifiés par cette procédure. La multiplication d'une matrice par un scalaire revient à multiplier chaque élément de la matrice par ce scalaire.
5. Écrire une fonction `int *multvect(MatriceCreuse_t M, int *v)` qui réalise la multiplication matrice-vecteur de `M` par `v`. Cette fonction alloue le vecteur résultat et renvoie un pointeur sur son premier élément. Comme la matrice est carrée, la taille de `v` et celle du vecteur résultat sont égales à la taille des dimensions de la matrice. Soit n cette taille, le vecteur résultat R est tel que $R[i] = \sum_{j=0}^{n} M[i][j] \times v[j]$ pour $0 \leq i < n$.
6. Donnez la définition d'une fonction `MatriceCreuse_t convert(int **M, int n)` qui convertit une matrice carrée pleine `M` de taille `n` et codée par un tableau bidimentionnel en une matrice creuse et renvoie le résultat. Cette fonction réalisera toutes les allocations dynamiques nécessaires.
   Écrire une fonction `MatriceCreuse_t transpose(MatriceCreuse_t M)` qui calcule la transposée de `M` et renvoie le résultat. La transposée d'une matrice carrée est une matrice carrée de même taille où lignes et colonnes ont été interverties.

Indication : Vous pouvez utiliser les fonctions classiques `void * malloc(int); void free(void *) ;`

Exercice 3 — Arbre binaire : création et destruction.

On se propose de donner du code permettant d'implanter une certaine sorte arbre binaire. L'exercice porte sur la construction et l'utilisation de type ainsi que l'usage des fonctions `void * malloc(int)` et `void free(void *)`. Seule la toute dernière question nécessite un peu d'algorithmique simple.

Nos arbres sont des structures de donnée constituées de cellules chacune constituée par :

— deux pointeurs d'identificateur `fillegauche` et `filledroite` pointant sur une cellule ou sur `NULL` si la cellule n'a ni fille gauche ni fille droit ;
— une structure (au sens C du terme) de type `data_t` et d'identificateur `data` ;

Questions.

1. Donner la déclaration du type `cellule_t` qui représente une cellule.
2. Donner la définition d'une fonction d'identificateur `CreerCellule` qui :
   — prend en paramètre un objet de type `data_t` ;
   — alloue sur le tas l'espace mémoire nécessaire à un objet de type `cellule_t` pour contenir les données passées en paramètre ;
   — affecte à `NULL` les pointeurs de cellule ;
   — affecte le champs `data` de cet objet grâce aux paramètres de la fonction ;
   — retourne un pointeur sur l'objet ainsi créé sur le tas.
3. Donner la définition d'une fonction de prototype :

   ```
   void DetruireCellule(cellule_t **) ;
   ```

   qui permet de désallouer l'espace mémoire associé tout affectant à `NULL` au pointeur pointé par le paramètre de cette fonction.
4. Donner la définition d'une fonction d'identificateur `DetruireArbre` qui prend en paramètre un pointeur sur un pointeur sur une cellule (la racine d'un arbre), retourne le nombre de cellules constituant l'arbre associé et détruit ce dernier. Les pointeurs référencant l'arbre doivent être mis à `NULL`.