Les listes chainées

François Delbot

18 septembre 2015

1 Définition d'une liste chainée (1 points)

`Prototype de la fonction :`

Définissez un nouveau type de donnée « struct chaine » permettant d'enregistrer un entier, nommé « val » ainsi qu'un pointeur vers une structure du même type nommé « next ». Renommez ensuite ce nouveau type en « liste ».

2 Création d'un élément de la liste (1 points)

`Prototype de la fonction : liste * creation_maillon(int n);`

Ecrire une fonction qui attend un entier en argument, puis alloue dynamiquement un element de type liste, affecte la valeur passée en argument au champs *val* et initialise le champs *next* à NULL. La fonction doit retourner l'adresse de cet element.

3 Liste vide ? (1 points)

`Prototype de la fonction : int liste_chainee_vide(liste *l);`

Ecrire une fonction qui accepte en argument un pointeur vers une liste et retourne 0 si cette liste est vide, 1 sinon.

4 Taille d'une liste chainée (1 points)

`Prototype de la fonction : int taille_liste_chainee(liste *maliste);`

Ecrire une fonction qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste (nommé maliste) et retourne le nombre d'éléments qui la constituent. Une liste vide aura une taille de 0.

5 Ajout en tête (1 points)

`Prototype de la fonction : liste * ajout_tete_liste_chainee(liste *maillon, liste *maliste);`

Ecrire une fonction qui accepte en argument un pointeur vers un element de type liste (*maillon*), ainsi qu'un pointeur vers une liste chainée constituée d'éléments de type liste (*maliste*) et renvoie une liste chainée ayant comme premier élément *maillon* et dont la suite est constituée par la liste *maliste*.

6 Affichage d'une liste chainée (1 points)

`Prototype de la fonction : void affichage_liste_chainee(liste *maliste);`

Ecrire une fonction qui affiche la valeur du champs *val* pour chaque élément d'une liste chainée constituée d'éléments de type liste. L'affichage, sans espace ni saut de ligne doit suivre le format suivant : $val_1 \rightarrow val_2 \rightarrow val_3 \rightarrow \ldots \rightarrow val_n$, avec val_i la valeur du champs *val* du i_{eme} élément de la liste chainée.

7 affichage récursif d'une liste chainée (1 points)

`Prototype de la fonction : void affichage_liste_chainee_rec(liste * maliste);`

Ecrire une fonction récursive qui affiche la valeur du champs *val* pour chaque élément d'une liste chainée constituée d'éléments de type liste. L'affichage, qui doit se faire dans l'ordre inverse des éléments de la liste, sans espace ni saut de ligne doit suivre le format suivant : $val_n \leftarrow \ldots \leftarrow val_3 \leftarrow val_2 \leftarrow val_1$, avec val_i la valeur du champs *val* du i_{eme} élément de la liste chainée.

8 Suppression en tête (1 points)

`Prototype de la fonction : liste * supprimer_tete_liste_chainee(liste *maliste);`

Ecrire une fonction qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste (*maliste*) et supprime le premier maillon de cette chaine (n'oubliez pas la libération de la mémoire de ce maillon) puis retourne un pointeur vers le premier element du reste de la liste. Si le reste de la liste est vide, retourner $NULL$.

9 Ajout en queue (2 points)

`Prototype de la fonction : liste * ajout_queue_liste_chainee(liste *maillon, liste *maliste);`

Ecrire une fonction qui accepte en argument un pointeur vers un element de type liste (*maillon*), ainsi qu'un pointeur vers une liste chainée constituée d'éléments de type liste (*maliste*). Cette fonction doit ajouter l'élément maillon à la fin de *maliste* et retourner l'adresse du premier élément de cette liste.

10 Ajout en queue (version récursive) (1 points)

`Prototype de la fonction : liste * ajout_queue_liste_chainee_rec(liste *maillon, liste *maliste);`

Ecrire une fonction récursive qui accepte en argument un pointeur vers un element de type liste (*maillon*), ainsi qu'un pointeur vers une liste chainée constituée d'éléments de type liste (*maliste*). Cette fonction doit ajouter l'élément maillon à la fin de *maliste* et retourner l'adresse du premier élément de cette liste.

11 Concaténation de deux listes chainées (1 points)

`Prototype de la fonction : liste * fusion_listes_chainees(liste *liste1,liste *liste2);`

Ecrire une fonction qui accepte en argument deux pointeurs vers des listes chainées constituées d'éléments de type liste (*liste*1 et *liste*2). Cette fonction doit retourner un pointeur vers le premier élément de la liste résultant de la concaténation de ces deux listes (*liste*1 puis *liste*2).

12 Suppression en queue (1 points)

`Prototype de la fonction : liste * suppression_queue_liste_chainee(liste *maliste);`

Ecrire une fonction qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste (*maliste*) et supprime le dernier maillon de cette chaine (n'oubliez pas la libération de la mémoire de ce maillon) et retourne un pointeur vers le premier element de la liste obtenue. Si la liste obtenue est vide, retourner $NULL$.

13 Insertion triée (1 points)

`Prototype de la fonction : liste * insertion_triee_liste_chainee(liste *maliste, liste *maillon);`

Ecrire une fonction qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste (*maliste*) supposée triée dans l'ordre croissant par la valeur contenue dans le champs *val*, ainsi qu'un pointeur vers un élément de type liste (*maillon*) et insère l'élément *maillon* dans *maliste* de sorte à ce que la liste chainée qui en résulte soit triée. Cette fonction doit également retourner un pointeur vers le premier élément de cette liste.

14 Suppression de la première occurence (1 points)

`Prototype de la fonction : liste * suppression_element_liste_chainee(liste *maliste, int nb);`

Ecrire une fonction qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste (*maliste*) ainsi qu'un entier *nb*. Cette fonction doit supprimer le premier élément de la liste dont la valeur du champs *val* vaut *nb*, et retourne un pointeur vers le premier élément de la liste ainsi modifiée.

15 Suppression de la première occurence (version récursive) (2 points)

`Prototype de la fonction : liste * suppression_element_liste_chainee_rec(liste *maliste, int nb);`

Ecrire une fonction récursive qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste (*maliste*) ainsi qu'un entier *nb*. Cette fonction doit supprimer le premier élément de la liste dont la valeur du champs *val* vaut *nb*, et retourne un pointeur vers le premier élément de la liste ainsi modifiée.

16 Liberée, delivrée... (1 points)

`Prototype de la fonction : void liberation_liste_chainee(liste *maliste);`

Ecrire une fonction qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste (*maliste*) et libère la mémoire de tous les éléments qui la composent.

17 libérée, délivrée, mais pas pareil... (2 points)

`Prototype de la fonction : void liberation_liste_chainee_rec(liste *maliste);`

Ecrire une fonction récursive qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste (*maliste*) et libère la mémoire de tous les éléments qui la composent.