Cours d'algorithmique et langage C

TDADS1140 – Listes Chaînées Bilatères

Réalisez l'implémentation des fonctions proposées en commençant par celles qui vous semblent les plus utiles puis les plus simples, pour finir sur les fonctions plus complexes comme celle de la permutation de deux maillons.

Une fonction peut en utiliser une autre si cela n'augmente pas la complexité algorithmique de façon sensible.

Les fonctions de type **int** renvoient un code d'erreur.

```
// LISTES BILATERES ************************************************

typedef struct DoubleLinkedListElem {
	int data;
	struct DoubleLinkedListElem *previous;
	struct DoubleLinkedListElem *next;
} DoubleLinkedListElem;

typedef struct DoubleLinkedList {
	DoubleLinkedListElem *head;
	int size;
	DoubleLinkedListElem *tail;
} DoubleLinkedList;


// création d'une nouvelle liste chaînée bilatère vide
// cette fonction renvoie un pointeur sur la nouvelle structure liste chaînée bilatère vide
DoubleLinkedList *newDoubleLinkedList();

// instanciation (création) d'un élément (maillon) à insérer dans une liste chaînée bilatère
// avec stockage de la donnée value dans l'élément de liste
DoubleLinkedListElem  *NewDoubleLinkedListItem(int value);
```

```c
// affichage des éléments d'une liste chaînée bilatère en commençant par la tête
int DisplayDoubleList(DoubleLinkedList *list);

// affichage des éléments d'une liste chaînée bilatère en commençant par la queue (Reverse)
int RevDisplayDoubleList(DoubleLinkedList *list);

// insertion d'un élément en queue de liste chaînée bilatère
int insertItemAtDoubleLinkedListTail(DoubleLinkedList *list, DoubleLinkedListElem *newItem);

// insertion d'un élément en tête de liste chaînée bilatère
int insertItemAtDoubleLinkedListHead(DoubleLinkedList *list, DoubleLinkedListElem *newItem);

// renvoie le nombre d'élément contenus dans la liste
int getDoubleLinkedListSize(DoubleLinkedList *list);

// insertion d'un élément après un autre élément
int insertItemAfterItem(DoubleLinkedList *list, DoubleLinkedListElem *item, DoubleLinkedListElem *newItem);

// insertion d'un élément avant un autre élément
int insertItemBeforeItem(DoubleLinkedList *list, DoubleLinkedListElem *item, DoubleLinkedListElem *newItem);

// destruction de tous les éléments d'une liste chaînée
int emptyDoubleLinkedList(DoubleLinkedList *list);

// suppression et destruction d'un élément de liste chainée
int deleteDoubleLinkedListItem(DoubleLinkedList *list, DoubleLinkedListElem *item);

// obtention d'un pointeur sur le prochain élément contenant "value", en partant de l'élément "item" et en
// allant soit vers la tête (UP), soit vers la queue (DOWN)
DoubleLinkedListElem * getNextDoubleLinkedListItem(DoubleLinkedList *list, DoubleLinkedListElem *item, int value, int sens);

// déplacement d'un élément après un autre élément
int moveDoubleLinkedListItemAfterItem(DoubleLinkedList *list, DoubleLinkedListElem *item, DoubleLinkedListElem *itemDest);

// permutation de deux éléments de la liste chainée
int swapDoubleLinkedListItem(DoubleLinkedList *list, DoubleLinkedListElem *itemA, DoubleLinkedListElem *itemB);
// trier au mieux
int sortDoubleLinkedList(DoubleLinkedList *list);
```

```c
// implémentation du tri bulles, pas très performant ...
int sortBublesDoubleLinkedList(DoubleLinkedList *list);


// modifie la valeur d'un élément
int setValueOfDoubleLinkedListItem(DoubleLinkedList *list, DoubleLinkedListElem *item, int value);
// obtient la valeur d'un élément
int getValueOfDoubleLinkedListItem(DoubleLinkedList *list, DoubleLinkedListElem *item, int *value);
// crée une nouvelle liste chaînée avec les éléments dont la valeur est comprise entre min et max
DoubleLinkedList * extractRangedValueFromNewDoubleLinkedList(DoubleLinkedList *list, int Min, int Max);
// clonage d'une liste chaînée (création d'une nouvelle liste qui contient la copie de tous les éléments)
DoubleLinkedList *cloneDoubleLinkedList(DoubleLinkedList *list);
// greffe d'une liste dans une autre liste à la suite de l'élément item,
// à l'issue de l'opération la liste greffée n'existe plus
int graftDoubleLinkedListAfterItem(DoubleLinkedList *list, DoubleLinkedListElem *item, DoubleLinkedList graftList);


// fonctions qui manipulent la liste en utilisant l'index (no d'ordre dans la liste) des Items
// renvoie EXIT_FAILURE ou NULL s'il n'y a pas d'élément en nième position

// renvoie un pointeur sur le nième élément de la liste
DoubleLinkedListElem *getItemAtIndexFromDoubleLinkedList(DoubleLinkedList *list, int index);
// insère un nouvel élément contenant "value" en nième position,
int insertItemAtIndexIntoLinkedList(DoubleLinkedList *list, int index, int value);
```