NF16
TP3

Listes chaînées

Printemps 2012

Objectif :

L'objectif de ce TP est de se familiariser avec les opérations les plus courantes des listes chaînées (création d'une liste, insertion de nœuds, suppression de nœuds, recherche d'information) et de programmer une petite application de gestion utilisant les listes chaînées.

Le développement :

Les listes chaînées sont des structures de données à accès indirect. Elles permettent de construire une suite d'éléments de même types ou plus rarement de types différents. L'avantage des listes est le fait qu'elles soient dynamiques. En effet, on alloue dynamiquement de la mémoire pour chaque élément de la chaîne, ce qui permet de gérer utilement les ressources en fonction des variations de la taille de la liste. Chaque élément, appelés *noeuds* ou *cellules*, contient une référence au prochain élément de la liste. Le deuxième avantage offert est la manipulation facile des éléments dans la liste (ajout, retrait,…). Pour simplifier les algorithmes, on utilise des listes chaînées circulaire dans lesquelles il y a un nœud particulier en début et fin de liste (appelé *sentinelle*). Ceci évite de devoir traiter de manière particulière le premier et dernier élément de la liste, puisque chaque nœud de la liste possède toujours un précédent et un suivant.

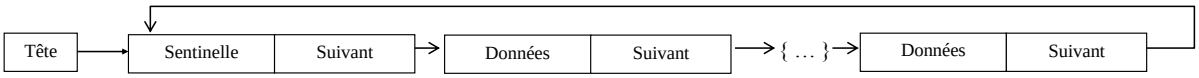

Figure 1. Liste circulaire avec sentinelle.

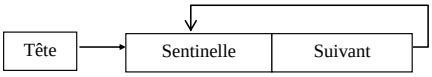

Figure 2. Liste circulaire vide.

L'application :

Il s'agit de développer une application simplifiée pour la gestion de stock d'un magasin. Le stock se compose d'un ensemble d'articles. Chaque article est caractérisé au minimum par :

- son identifiant unique ;
- son libellé;
- son prix unitaire ;
- sa quantité ;
- sa catégorie (alimentaire, boisson, hygiène, arts ménagers, textile, papeterie, etc.).

On pourra aussi gérer des requêtes (commandes clients ou approvisionnement du stock) caractérisées par :

- l'identifiant de l'article ;
- la quantité à ajouter ou à retirer selon le type de requête.

Listes chaînées

Printemps 2012

Structures de données :

Les structures suivantes représentent un *Article* et une *Requête* et contiennent les champs nécessaires pour stocker les informations mentionnées ci-dessus.

```
typedef struct Art
{
   int id;                    // identificateur unique d'un article > 0
   char libelle[50];          // libellé de l'article (son nom)
   float PU;                  // prix unitaire > 0
   int qte;                   // quantité de l'article présente dans le stock > = 0
   char categorie[50];  // catégorie de l'article (alimentaire, hygiène, textile, papeterie, etc.)
   struct Art * nextArticle;
}Article;

typedef Article * ListeArticles;

typedef struct Req
{
   int idArticle;          // identificateur de l'article
   int qte;                // quantité à retirer ou à ajouter dans le stock correspondant à idArticle
   struct Req * nextRequete;
}Requete ;

typedef Requete * ListeRequetes;
```

Il existe trois listes dans cette base de données :

- Liste_stock_dispo : contient les articles disponibles
- Liste_stock_epuise : contient les articles à renouveler
- Liste_requetes : l'ensemble des mises à jour à effectuer sur les articles du stock

Fonctions à réaliser :

Créer les fonctions suivantes pour la manipulation des listes chaînées :

1. *void AjoutArticle(ListeArticles l, Article * a)* : cette fonction insère un article dans la bonne position (le stock est trié par prix unitaire dans l'ordre décroissant).

2. *void AfficheStock(ListeArticles l)* : cette fonction affiche une liste d'articles.

Exemple d'affichage :

```
-------------------------------------------------------------------------------
| Identifiant |    Libellé          | Prix Unitaire |  Quantité  | Catégorie   |
-------------------------------------------------------------------------------
| 15          |   Stylo à plume     |          2,50 |         25 |  Papeterie  |
| 25          | Veste courte        |         10,00 |         50 |  Textile    |
-------------------------------------------------------------------------------
```

3. *void SupprimeStock(ListeArticles l)* : cette fonction efface tous les articles de la liste (la mémoire utilisée par ces éléments doit être libérée).

4. *void SupprimeArticle(ListeArticles l,* ***int*** *id_Article)* : cette fonction supprime l'article ayant l'identifiant *id_Article*.

5. *ListeArticles RechercheArticleParCritere(ListeArticles l,* ***int*** *id,* ***char***** libelle,* ***char***** categorie)* : cette fonction affiche une liste d'articles selon les critères passés en paramètre. Elle prend en paramètre l'identifiant, le libellé, et/ou la catégorie. On utilise la valeur -1 pour ignorer le paramètre id lors de la recherche et le mot clé "IGNORE" pour les deux autres paramètres (libelle et catégorie).

> **Exemples :**
>
> *RechercheArticleParCritere(Liste_stock_dispo, 12, "stylo" , "papeterie")*
> Retourne l'article ayant l'identifiant 12, le libellé stylo et de la catégorie papeterie.
>
> *RechercheArticleParCritere(Liste_stock_dispo, -1, "*IGNORE" *, "papeterie")*
> Retourne tous les articles appartenant à la catégorie papeterie.

6. *ListeArticles RechercheParQuantite(Liste_stock_dispo,* ***int*** *qte_min,* ***int*** *qte_max)* : cette fonction affiche tous les articles dont la quantité est comprise entre les bornes passées en paramètres : [*qte_min*, *qte_max*]

7. *ListeArticles RechercheParPrixUnitaire (Liste_stock_dispo,* ***int*** *prix min,* ***int*** *prix_max)* : cette fonction affiche tous les articles dont le prix unitaire est compris entre les bornes passées en paramètres : [*prix_min*, *prix_max*]

8. *void MiseAJourArticle(****int*** *id,* ***int*** *quantite)* : cette fonction met à jour la quantité d'un article donné. Dans le cas où la quantité devient nulle, l'article est supprimé de la liste du stock disponible (Liste_stock_dispo) et il est inséré dans la liste du stock épuisé (Liste_stock_epuise).

9. *void VendreArticles(ListeRequetes r)* : cette fonction permet de diminuer le stock d'une liste d'articles. Elle prend en paramètre une liste de requêtes.

10. *void ApprovisionnerMagasin(ListeRequetes r)* : cette fonction permet d'augmenter le stock d'une liste d'articles. Elle prend en paramètre une liste de requêtes.

> **Remarque 1 :** Les fonctions 9 et 10 feront appel à la fonction *MiseAJourArticle()*.

11. *void LireFichier(****char***** nomfichier)* : cette fonction ouvre un fichier dont le nom est spécifié en paramètre et crée les listes *Liste_stock_dispo* et *Liste_stock_epuise.*

> **Remarque 2 :** La base de données est un fichier de type texte où les enregistrements ne sont pas triés. Chaque ligne contient les caractéristiques d'un article : id, libellé, prix unitaire, quantité et catégorie. Les champs sont séparés par une tabulation.

ID	LIBELLE	PRIX	QUANTITE	CATEGORIE
12	stylo	2,5	40	Papeterie
15	valise	10	10	Divers
1	jupe	5	25	Textile

Interface :

L'utilisateur (un commerçant, par exemple) veut connaître à tout moment l'état du stock de son magasin (quantité restantes dans une catégorie d'article, articles à approvisionner, …). Il pourra aussi effectuer des opérations de commandes clients et de réapprovisionnement du stock.
A l'aide des fonctions ci-dessus, écrire un programme proposant un menu avec les fonctionnalités suivantes :

1. Lecture des données à partir d'un fichier
2. Ajout d'un nouvel article non existant dans le magasin à partir du clavier
3. Affichage des caractéristiques des articles du stock
    - Stock disponible
    - Stock épuisé
    - Tout le stock
4. Suppression d'un article
5. Recherches d'articles
    - Recherche par identifiant, libellé et/ou catégorie
    - Recherche par quantité
    - Recherche par prix unitaire
6. Gestion des commandes clients (saisies au clavier)
7. Approvisionner le stock (saisies au clavier)
8. Suppression du stock
9. Quitter

Notes :

Le programme doit être composé d'au moins 3 fichiers :

tp3.h : contient les constantes globales, les types définis et les prototypes de toutes les fonctions. Chaque fonction doit être déclarée avec un commentaire expliquant à quoi elle sert, quels sont les paramètres d'entrées et quelle(s) sont les valeur(s) de sorties.

tp3.c : contient le corps des fonctions déclarées dans tp3.h. Il doit inclure l'instruction *#include "tp3.h"*.

tp3_main.c : contient uniquement la fonction *main()*. L'instruction *#include "tp3.h"* doit y être aussi.

- D'autres fonctions seront nécessaires au bon fonctionnement de l'ensemble des modes de traitement demandés. A vous de les identifier et de les développer.
- Un rapport d'au plus 4 pages est à rendre, contenant une introduction, les choix de programmation, le calcul de complexité des principaux algorithmes et une conclusion, etc (en reprenant la problématique, en spécifiant les difficultés rencontrées, les solutions apportées, les fonctions rajoutées).
- Le code source rigoureusement commenté et sécurisé, ainsi qu'un exécutable seront transmis aux chargés de TPs.