TP1 - 3h
Tableaux dynamiques vs Listes chainées

Romain Marie

2024-2025

3iL INGENIEURS
</les sciences informatiques>

Dans ce TP, vous allez mettre en place et comparer plusieurs structures de données permettant de représenter et manipuler les stocks d'un supermarché.

Les objectifs principaux du TP sont :

- de (re)prendre en main les fondamentaux du langage C
- d'implémenter et tester les fonctions élémentaires d'un tableau dynamique
- d'implémenter et tester les fonctions élémentaires d'une liste chainée
- de retrouver par l'expérimentation les complexités algorithmiques des opérations de bases pour chacune des deux structures

Remarque : si vous le souhaitez, vous pouvez réaliser ce TP en binôme.

Remarque 2 : certaines questions, marquées d'une étoile peuvent être sautées si vous vous sentez en retard dans le TP

1 Prise en main de l'existant

L'archive qui vous est fournie contient le squelette du projet que vous allez devoir réaliser lors de ce TP. Il est structuré comme suit :

- Le fichier main.c contient le point d'entrée du programme, et devra donc être ajusté au fil du TP pour mettre en oeuvre les tests et comparaisons qui vous sont demandées.
- les fichiers utils.h et utils.c contiennent un ensemble de fonctions déjà écrites qui vous seront utiles lors du TP. **Vous n'avez aucune raison d'apporter des modifications à ces fichiers.**
- Le fichier article.h est pour l'instant presque vide. Pas de panique, c'est le premier que vous remplirez.
- Les fichiers stockTableau.h, stockTableau.c, stockListe.h et stockListe.c contiennent les prototypes et les squelettes des fonctions que vous allez devoir écrire. **C'est ici que le gros du travail se passera**.

1. Première compilation
   (a) Dans le fichier main.c, complétez le point d'entrée pour qu'il affiche le message de votre choix. Vous utiliserez pour cela la fonction *printf()* après avoir inclus *stdio.h*
   (b) Dans un terminal, placez-vous dans le répertoire de votre projet, puis compilez en utilisant gcc : **gcc -o tp1 main.c**
   (c) Exécutez votre programme, et vérifiez que le message s'affiche.

2. Mise en place et manipulation de la structure Article
   (a) Dans le fichier article.h, définissez la structure **Article**, en respectant les champs suivants :
      - **id** : identifiant (unique) de l'article
      - **nom** : label de l'article (100 caractères max)
      - **prix** : prix d'achat de l'article
      - **stock** : quantité actuellement en stock dans le magasin
   (b) Dans le main, créez un Article dans la pile, et un autre dans le tas.
   (c) Toujours dans le main, en utilisant la fonction *chargerArticles()* qui vous est fournie, remplissez ces deux articles avec le fichier "2articles.csv".
   (d) Toujours dans le main, présentez avec un bel affichage ces deux articles.
   (e) Dans un terminal, compilez en utilisant gcc (sans oublier le fichier utils.c qui fait maintenant partie des sources)
   (f) Exécutez et vérifier que vous obtenez le résultat escompté.

2 Tableau dynamique

2 articles c'est bien, mais ça fait pas beaucoup de choix dans les rayons. Il est donc temps de considérer un vrai catalogue d'articles, que nous allons pour l'instant représenter par un tableau dynamique dont la taille évolue en fonction des besoins.

1. Mise en place du tableau dynamique
   (a) Dans le fichier *stockTableau.h*, définissez la structure **StockTableau** en la dotant des champs suivants :
      - **articles** chargé de recueillir les différents articles
      - **nbArticles** qui désigne le nombre d'articles actuellement saisis
      - **tailleMax** qui correspond à la capacité courante du tableau
   (b) Dans le fichier *stockTableau.c*, écrivez le contenu des fonctions *initStock()* et *libererMemoire()*, chargés respectivement d'allouer et de libérer la mémoire d'un StockTableau.
   (c) Toujours dans le fichier *stockTableau.c*, écrivez le contenu de la fonction *ajouterArticle()* qui ajoute un article au tableau. On supposera pour l'instant que :
      - l'article ajouté n'est pas déjà présent dans le tableau (tant pis s'il y a redondance)
      - si le tableau est plein, il est redimensionné en lui ajoutant seulement la case dont on a besoin
   (d) En utilisant les fonctions précédentes, et en vous inspirant de *chargerArticles()*, complétez la fonction *chargerFichier()* qui se trouve dans *stockTableau.c*.
   (e) Ecrivez une fonction *afficherStockTableau()* qui présente (joliment) un StockTableau.

2. Validation des briques de base
   (a) Dans le point d'entrée, utilisez le fichier "10articles.csv" pour créer et remplir un StockTableau

(b) Vérifier que tout va bien en affichant votre stock

(c) Pensez à libérer la mémoire en fin de programme !

(d) Après avoir compilé (en étant vigilant sur la liste des sources), vérifiez que tout se passe comme prévu à l'exécution.

3. Un peu d'algo !

(a) Dans le fichier *stockTableau.c*, ajoutez une fonction *retirerArticle()* qui, à partir de l'identifiant d'un article, retire 1 à son stock. Dans le cas où le stock atteint 0, l'article est retiré du tableau. La fonction renverra 0 si tout s'est bien passé, et 1 si l'article n'a pas été trouvé. **Attention : l'identifiant d'un article n'est pas égal à sa position dans le tableau**

(b) Dans le fichier *stockTableau.c*, ajoutez une fonction *quantiteEnStock()* qui, à partir du nom d'un article, renvoie combien sont en stock. Vous pourrez utiliser la fonction *strcmp()* de la librairie *string.h*. **On suppose pour cette question que la valeur à retourner correspond au stock de la première case du tableau qui porte le nom de l'article. Si le coeur vous en dit, rien ne vous interdit de faire le cumul pour tous les doublons**

(c) * Dans le fichier *stockTableau.c*, ajoutez une fonction *fusionnerDoublons()* qui fusionne en une seule case celles qui contiennent un article portant le même nom.

4. Premières mesures

(a) En utilisant avec attention la fonction *clock()* de la librairie *time.h*, mesurez successivement le temps d'exécution de la fonction *afficherStockTableau()* pour les fichiers "10articles.csv", "100articles.csv", et "1000articles.csv". Déduisez-en la complexité temporelle de cette fonction.

(b) Procédez à la même étude sur la fonction *chargerFichier()*, mais en utilisant cette fois les fichiers "1000articles.csv", "10000articles.csv" et "100000articles.csv". Vous considérerez une tailleMax initiale égale à 10. **Les résultats devraient varier significativement d'une exécution à l'autre, savez-vous pourquoi ?**

(c) Modifiez le comportement de la fonction *ajouterArticle()* pour qu'à chaque dépassement du tableau, sa tailleMax soit multipliée par 1.3 (au lieu de simplement l'incrémenter).

i. Mesurez maintenant le temps que met *chargerFichier()* pour le fichier "100000articles.csv".

ii. Sachant qu'un char est stocké sur 1 octet, qu'un int et qu'un float sont stockés sur 4 octets, et qu'un double est stocké sur 8 octets, quantifiez l'espace perdu

(d) Modifiez le comportement de la fonction *ajouterArticle()* pour qu'à chaque dépassement du tableau, sa tailleMax soit doublée.

i. Mesurez maintenant le temps que met *chargerFichier()* pour le fichier "100000articles.csv".

ii. Sachant qu'un char est stocké sur 1 octet, qu'un int et qu'un float sont stockés sur 4 octets, et qu'un double est stocké sur 8 octets, quantifiez l'espace perdu

3 Liste chainée

L'objectif du TP étant de comparer les deux approches, notre rigueur scientifique nous impose maintenant d'implémenter les mêmes fonctionnalités, mais cette fois à partir d'une liste chainée. Faites attention, nous n'en avons pas fini avec l'utilisation du tableau. Gardez donc dans un coin tout ce qui a été fait jusqu'à maintenant.

1. Mise en place de la liste chainée

(a) Dans le fichier *stockListe.h*, définissez la structure **NoeudArticle** en la dotant des champs suivants :

- **next** chargé de pointer vers le NoeudArticle suivant

- **article** qui contient l'article stocké dans le noeud

(b) Dans le même fichier, définissez la structure **StockListe**, dont le seul champ sera un pointeur vers la tête de la liste.

(c) Dans le fichier *stockListe.c*, écrivez le contenu des fonctions *initStock()* et *libererMemoire()*, chargés respectivement d'allouer et de libérer la mémoire d'un StockListe.

(d) Toujours dans le fichier *stockListe.c*, écrivez le contenu de la fonction *ajouterArticle()* qui ajoute un article à la liste.

(e) En utilisant les fonctions précédentes, et en vous inspirant de *chargerArticles()*, complétez la fonction *chargerFichier()* qui se trouve dans *stockListe.c*.

(f) Ecrivez une fonction *afficherStockListe()* qui présente (joliment) un StockListe.

2. Validation des briques de base

(a) Dans le point d'entrée, utilisez le fichier "10articles.csv" pour créer et remplir un StockListe

(b) Vérifier que tout va bien en affichant votre stock

(c) Pensez à libérer la mémoire en fin de programme !

(d) Après avoir compilé (en étant vigilant sur la liste des sources), vérifiez que tout se passe comme prévu à l'exécution.

3. Un peu d'algo !

(a) Dans le fichier *stockListe.c*, ajoutez une fonction *retirerArticle()* qui, à partir de l'identifiant d'un article, retire 1 à son stock. Dans le cas où le stock atteint 0, l'article est retiré de la liste. La fonction renverra 0 si tout s'est bien passé, et 1 si l'article n'a pas été trouvé. **Attention : l'identifiant d'un article n'a rien à voir avec sa position dans la liste.**

(b) Dans le fichier *stockListe.c*, ajoutez une fonction *quantiteEnStock()* qui, à partir du nom d'un article, renvoie combien sont en stock. Vous pourrez utiliser la fonction *strcmp()* de la librairie *string.h*. **On suppose pour cette question que la valeur à retourner correspond au stock du premier noeud de la liste qui porte le nom de l'article. Si le coeur vous en dit, rien ne vous interdit de faire le cumul pour tous les doublons**

(c) * Dans le fichier *stockTableau.c*, ajoutez une fonction *fusionnerDoublons()* qui fusionne en un seul noeud tous ceux qui contiennent un article portant le même nom.

4. Encore des mesures

(a) Mesurez successivement le temps d'exécution de la fonction *afficherStockListe()* pour les fichiers "10articles.csv", "100articles.csv", et "1000articles.csv". Vérifiez qu'ils sont du même ordre de grandeur que pour la version tableau dynamique.

(b) Après avoir pris quelques minutes pour évaluer les complexités temporelles des différentes fonctions implémentées (pour les deux approches), vérifiez expérimentalement. Vous considérerez a minima :

- l'ajout d'un article
- la suppression d'un article
- la recherche d'un article (quantité en stock)

4 Pour aller plus loin

Les tableaux dynamiques et les listes chainées sont les représentations de base en structure de données. Elles ne sont en réalité qu'assez peu spécifiques, et n'ont donc pas d'avantage dédiés à une utilisation particulière. Dans le contexte d'une gestion de stock, l'opération la plus courante est la recherche d'un article en particulier, qui peut être réalisée en O(log(n)) dans le cas d'une structure de donnée (tableau ou liste chainée) triée. A vous de jouer ...