TP5 : Structures de données

Préambule

L'objectif de cette séance est de se familiariser avec les structures de données vus en cours. À la fin de ce TP, vous devriez être capable de mettre en œuvre :

- un tableau dynamique et ses opérations,
- une liste chaînée avec des éléments alloués individuellement.

Quelques informations importantes :

- Le langage de programmation à utiliser est le C.
- Ce TP utilise les signaux (pour interrompre les tris lorsqu'ils dépassent une certaine limite de temps), il doit être fait sous Linux.
- Des outils pour tester les fonctions sont fournis dans le fichier zip `algo-tp05-code.zip`.
- Pour chaque exercice, exécutez les tests dans les fichiers fournis pour vérifier le code écrit.
- Il est fortement recommandé de **compiler dans le terminal** (en utilisant `make`).

1 Outils

Les fichiers suivants sont fournis (`tp05_code.zip`) et seront utilisés aussi pour le TP6 :

- `main.c` – le programme principal et les fonctions permettant de tester les structures
- `tests.c` et `tests.h` – les fonctions de test
- `tableau_double.h` – la structure et opérations pour le tableau dynamique
- `liste_double.h` – la structure et opérations pour la liste chaînée
- `liste_tableau_double.h` – la structure pour la liste chaînée sur un tableau (TP6)
- `util.c` et `util.h` – des utilitaires (remplissage de tableau, affichage de tableau, ...)
- `chrono.c` et `chrono.h` – des fonctions pour chronométrer et interrompre les exécutions
- `makefile` – un fichier permettant de créer l'exécutable en utilisant l'utilitaire make

Exercice 1 : Prise en main du programme.

1. Copiez tous ces fichiers dans un répertoire.
2. Placez-vous dans ce répertoire.
3. Lancez l'utilitaire `make` : `make`
4. Lancez le programme : `./main`

Le programme affiche alors la configuration actuelle, puis propose un menu :

```
--------------- Configuration actuelle --------------

Taille               : 10000
Limite de temps      : 2.000 s

        --- Menu ---
    1. Lancer les tests du tableau dynamique
    2. Lancer les tests de la liste chainee
    3. Lancer les tests de tri

    5. Changer le (n)ombre d'elements
    6. Changer la (l)imite de temps

    0. (Q)uitter le programme
```

La configuration de départ s'affiche et vous pouvez, sur un tableau de taille comme indiqué, lancer les tests de :

- tableau dynamique : `1`
- liste chaînée : `2`
- tri par fusion (TP6) : `3`

Lors de l'exécution, si le programme peut exécuter les tests dans le temps imparti, il affichera le temps effectivement mis. Sinon, il affichera un message indiquant qu'il n'a pas pu bien exécuter les tests dans le temps limite. Si le test trouve une erreur, un message vous l'indiquera.

Ainsi, si vous entrez `2` , vous obtenez (résultats sur ma machine, les temps peuvent évidemment différer) :

```
------------- Tests de la liste chainee -------------
Test de nouveau maillon...  Reussi
Test de insertion apres...  Reussi
Test de avancer...          Element 2500 incorrect
Erreur dans les tests!
```

Vous pouvez également modifier les paramètres de configuration :

- modifier le nombre d'éléments du tableau : `5` ou `n`
- modifier la limite de temps au delà de laquelle le tri s'interrompra : `6` ou `l`

Exercice 2 : Lecture des programmes fournis (*). Éditez les différents fichiers fournis et analysez-les. Il n'est pas demandé de comprendre le fichier chrono.c.

2 Les tableaux dynamiques

Les exercices suivants visent à compléter l'implémentation d'un tableau dynamique de réels (`double`).

Exercice 3 : Opérations basiques (*). Écrivez et testez les opérations `tbl_dbl_nouveau`, `tbl_dbl_redim` et `tbl_dbl_inserer`, en modifiant le code vu en cours pour opérer sur des réels.

Exercice 4 : Supprimer un élément (**). Écrivez et testez la fonction `tbl_dbl_supprimer` qui prend en entrée un pointeur vers `tbl_dbl` et une position i et supprime l'élément du tableau à la position i.

Exercice 5 : Redimensionnement après suppression (*). Modifiez la fonction pour qu'elle redimensionne le tableau si sa capacité est supérieure à quatre fois le nombre d'éléments. Dans ce cas, la fonction doit réduire la capacité à la moitié.

3 Les listes chaînées

L'objectif de ces exercices est d'implémenter une liste chaînée de réels.

Exercice 6 : Création et insertion simples (*). Vérifiez et testez `nouveau_maillon_dbl` et `liste_dbl_inserer_apres`, qui utilisent le code vu en cours.

Exercice 7 : Opération utiles (*). Écrivez et testez les fonctions suivantes :

1. Une fonction `liste_dbl_avancer`, qui prend en paramètre un pointeur vers un maillon m et un entier $i \geq 0$, et retourne un pointeur vers le maillon qui se trouve i places après m. Par exemple,
   - `liste_dbl_avancer(m, 0)== m`,
   - `liste_dbl_avancer(m, 1)== m->suiv`.
2. Une fonction `liste_dbl_longueur`, qui prend une liste en paramètre et renvoie sa longueur, c'est-à-dire le nombre d'éléments jusqu'à ce que `NULL` soit atteint.

Exercice 8 : Insertion (**). Écrivez et testez la fonction `liste_dbl_inserer_position` qui prend comme paramètres une liste `lst`, une position i et une valeur v, et insère un élément à la position i de `lst`, en décalant les éléments suivants.

Exercice 9 : Suppression (**). Écrivez et testez des fonctions de suppression :

1. `liste_dbl_supprimer_apres`, qui prend un pointeur vers un maillon et supprime l'élément suivant ;
2. `liste_dbl_supprimer_position`, qui prend une liste et une position et supprime l'élément de la liste à la position donnée.

Exercice 10 : Opérations de parcours (*). Écrivez et testez des fonctions :

1. `liste_dbl_afficher`, qui affiche les valeurs d'une liste donnée, et
2. `liste_dbl_liberer`, qui libère (avec `free`) tous les éléments d'une liste donnée.

4 Pour aller plus loin

Exercice 11 : Listes doublement chaînées (***). Modifiez la structure des listes chaînées (dans la section 3) pour ajouter un attribut `prec` qui stocke un pointeur sur l'élément précédent de la liste (pour le premier élément, on stocke le pointeur sur le dernier élément pour y accéder plus facilement).

Modifiez également les opérations et vérifiez que les insertions en fin de liste deviennent plus rapides.

Note: Une fonction `liste_dbl_inserer_fin` a été ajouté au fichier `tests.c`. Veuillez la déplacer dans le bon fichier (`liste_double.c`) avant de la modifier. Si besoin, vous pouvez modifier les noms des fonctions et structures dans la fonction `test_liste_fin`.

Exercice 12 : Tri rapide sur des listes (***). Implémentez le tri rapide sur des listes chaînées, de manière à obtenir une complexité dans le cas moyen de $\Theta(n \log n)$.