TP2 : Pointeurs sur fonctions

Buts : donner un aperçu
- des pointeurs sur fonctions
- du mécanisme de la programmation objet

1 Installation

1. Récupérez l'archive `TP_Ptr_Fonctions.tar.gz` à partir :
   - de ma page *Enseignements* à `http ://www-info.univ-lemans.fr/~jacob/enseignement.html` dans la rubrique *Programmation C*
   - du serveur de l'IUP à `/info/tmp/AnnexesTPL2SPI/TP_Ptr_Fonctions/TP_Ptr_Fonctions.tar.gz`
2. Décompressez la et désarchivez la.
3. Comme dans le TP précédent, normalement tous les programmes et modules peuvent être compilés par `make -f Makefile all`

2 Sujet

2.1 Cadre

Il faut concevoir dans ce TP un TDA Liste pour stocker des objets hétérogènes (pas de même type). Pour tester cela 3 objets différents vous sont proposés
- des objets de type `individu_t` ayant un prénom et un nom
- des objets de type `fraction_t` ayant un numérateur et un dénominateur
- des objets de type `string_t` qui sont des chaînes de caractères

Mais vous pouvez également en créer d'autres.
Le but sera donc de mettre ces objets dans une seule liste.

2.2 Affichage

Nous nous pencherons d'abord sur le parcours de la liste pour afficher tous ses éléments. Le TDA liste ne sait pas *a priori* quelle fonction doit être utilisée pour afficher chacun de ses éléments.
- si c'est un élément de type `individu_t` alors il faut appeler la fonction `individu_afficher`
- si c'est un élément de type `fraction_t` alors il faut appeler la fonction `fraction_afficher`

– si c'est un élément de type `string_t` alors il faut appeler la fonction `string_afficher`

Pour répondre à ce problème, chacun des objets de la liste contiendra un pointeur sur la fonction d'affichage qu'il faudra utiliser. La fonction d'affichage du TDA liste (`liste_afficher`) n'aura plus alors qu'à invoquer la fonction pointée.

2.3 Destruction

Ensuite nous nous pencherons sur la destruction de tous ses éléments, ainsi que la liste elle-même. Le problème et sa solution sont les mêmes que pour celui de l'affichage des éléments : chaque objet aura un deuxième pointeur sur une fonction qui est capable de le détruire (libérer sa place mémoire). La fonction pointée sera invoquée par la fonction de destruction du TDA liste (`liste_detruire`).

3 Étape 1 : création des objets

Pour l'objet `individu_t` :

1. Complétez sa définition (sa structure) dans `individu.h`
2. Compléter ses fonctions de création et de destruction dans `individu.c`
3. Testez ces fonctions et méthodes de cet objet par le programme `test_individu`. Il n'est pas non plus inutile de vérifier d'éventuelles fuites de mémoire par `valgrind`.

Faites de même avec les objets `fraction_t` et `string_t`

4 Étape 2

Programmez ensuite le TDA liste qui fera donc appel aux "méthodes" des 3 objets ci-dessus :

1. Complétez sa définition (sa structure) dans `liste.h`, en particulier le type `objet_t` des éléments de la liste
2. Compléter ses fonctions de création d'affichage et de destruction dans `liste.c`.
3. Testez de TDA par le programme `test_liste`.

5 Étape 3

Copier ensuite les fichiers
- `liste.h` en `liste_objet.h`
- `liste.c` en `liste_objet.c`
- `test_liste.c` en `test_liste_objet.c`

Modifiez ensuite les fichiers `liste_objet*` pour que le TDA liste soit lui-même un objet. Pour cela

1. Transformez toutes ses fonctions, a part
   - `liste_creer`
   - `liste_existe`

   en méthodes. C'est à dire que chaque fonction sera référencée par un pointeur de fonction dans l'objet `liste_t`.
2. Modifiez le programme `test_liste_objet.c` pour tester cette nouvelle version du TDA

6 Étape 4

Créer un cinquième type d'objet `personne_t` qui hérite de `individu_t`.
- l'affichage d'un objet de type `personne_t` fera appel à la fonction d'affichage d'un `individu_t`
- idem pour la création
- pour la destruction d'une `personne_t` on fera appel à une fonction qui détruit uniquement les champs de `individu_t`