Séance 5 -Structures

Le but de ce TP est de manipuler les structures en C. Comme pour les TP précédents, vous créerez trois fichiers : `fonctions.h`, `fonctions.c` et `main.c`.
Les exercices suivants sont indépendants les uns des autres. Ainsi, vous créerez dans le `main` une fonction de test par exercice. Dans la fonction `main()` figureront donc seulement les appels à ces fonctions de test, que vous pourrez commenter facilement pour ne tester qu'un exercice à chaque fois.
Pour récapituler :
- Dans le fichier `fonctions.h` se trouveront les signatures des fonctions qui vous seront demandées dans les exercices.
- Dans le fichier `fonctions.c` se trouveront au début les structures à définir, et ensuite les corps des fonctions déclarées dans le header.
- Dans le fichier `main.c` se trouveront une fonction par exercice (appelez les par exemple `exercice1()`, `exercice2()`...etc), ainsi que la fonction `main()` qui ne comportera que des appels à ces fonctions.

Ne pas oublier d'inclure `<stdio.h>`, `<stdlib.h>` et `"fonctions.h"` dans le fichier `main.c`, et d'inclure `<stdio.h>`, `<stdlib.h>`, `<stddef.h>` dans le fichier `fonctions.h` (et d'inclure `"fonctions.h"` dans le fichier `fonctions.c`).

Attention :
- **même si ce n'est pas demandé dans chaque exercice, veuillez à bien tester chacune de vos fonctions (dans les fonctions du main, cf précédemment).**
- **merci d'indenter correctement et de commenter votre code.**

Exercice 1 Pour représenter un nombre complexe, définir une structure `Complexe` qui contient deux flottants : la partie réelle et la partie imaginaire. Puis :
- écrire une fonction `creerComplexe` qui prend en entrée deux flottants et qui renvoie le nombre complexe associé (en utilisant la structure).
- écrire une fonction `afficheComplexe` qui prend en entrée un `Complexe`, ne retourne rien mais affiche le nombre complexe passé en paramètre. Par exemple l'affichage donnera "$2 + 5i$".
- écrire une fonction `additionComplexe` qui prend en entrée deux `Complexe` et qui retourne un `Complexe` qui est l'addition des deux nombres passés en paramètres.

Exercice 2 On souhaite représenter un polynôme par une structure. Definissez une structure `Polynome` contenant un tableau dynamique de flottants ainsi que sa taille (= degré du polynôme+1).
- Ecrire une fonction `creerPolynome` qui prend en entrée un tableau de flottants (et sa taille), et retourne le `Polynome` correspondant.
- Ecrire une fonction `afficherPolynome` qui prend en entrée un `Polynome` et l'affiche (elle ne retourne donc rien). Par exemple l'affichage donnera "$2x^3 + 0.5x - 2$".
- Ecrire une fonction `derive` qui prend en entrée un `Polynome`, et retourne sa dérivée (sous la forme d'un `Polynome`).
- Ecrire une fonction `evalPolynome` qui prend en entrée un `Polynome` p et un flottant x, et qui retourne l'évaluation de p en x : $p(x)$.

Exercice 3 Écrire une structure `Temps` qui comporte trois champs flottants : heure, minute et seconde.
- écrire une fonction `creerTemps` qui prend en entrée trois flottans et retourne la structure `Temps` correspondante.
- écrire une fonction `afficheTemps` qui prend en entrée un `Temps` et l'affiche (elle ne retourne rien).
- écrire une fonction `additionTemps` qui prend en entrée deux `Temps` et retourne l'addition des deux temps (pour les heures, faire seulement l'addition modulo 24, autrement dit on ne passe pas au jour suivant ou quoi que ce soit).
- Définir une fonction `compareTemps` prend en entrée deux `Temps` `t1` et `t2`, qui renvoie -1 si `t1<t2` ; 1 si `t1>t2` ; 0 si `t1=t2`.

On cherche maintenant à définir une structure `CD` contenant : un nombre de pistes et leurs durées respectives.
- Écrire la structure `CD`.
- Écrire une fonction `creerCD` qui prend en entrée un tableau de `temps` (et sa taille) et retourne la structure `CD` correspondante.
- Écrire une fonction `dureeTotale` qui prend en entrée un `CD` et retourne sa durée totale.

– Écrire une fonction `pistesSuperieuresA` qui prend en entrée un `CD` ainsi qu'un flottant t et qui retourne le nombre de pistes du CD dont la durée est supérieure à t secondes.

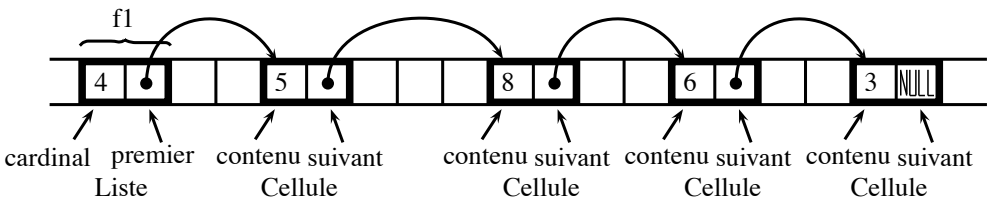
Exercice 4 Ecrire une structure `Point` qui représente un point dans le plan, et comporte donc trois champs : un identifiant (nom), sous forme d'un `char`, et deux entiers `x` et `y`.
– Ecrire une fonction `creerPoint` qui prend en entrée un `char` et deux `int` et retourne le `Point` correspondant.
– Ecrire une fonction `afficherPoint` qui prend un `Point` en entrée, ne retourne rien mais affiche ce point. On pourra par exemple afficher `"point B de coordonnées (8, 12)"`.
– Ecrire une fonction `distance` qui prend en entrée deux `Point` et retourne la distance euclidienne entre ces points.
En utilisant la structure précédente, définir une structure `LignePolygonale` contenant un tableau de `Point` ainsi que le nombre de points.
– Ecrire une fonction `creerLignePolygonale` qui ne prend aucun paramètre en entrée, et retourne une `LignePolygonale` vide.
– Ecrire une fonction d'affichage d'une `LignePolygonale`.
– Ecrire une fonction `ajoutPoint` qui prend en entrée une `LignePolygonale` et un `Point`, et ajoute ce point à la ligne. Cette fonction ne retourne rien, mais la ligne doit être modifiée (indication : vous pouvez utiliser un pointeur).
– Ecrire une fonction `longueurLigne` qui prend en entrée une `LignePolygonale` et retourne la longueur de cette ligne.
– Ecrire une fonction `creerCarre` qui prend en entrée un entier c et un point A, et qui retourne sous forme de `LignePolygonale` un carré de côté c et dont le coin supérieur gauche est A.

Exercice 5 On voudrait créer une structure de donnée de type liste en C. Pour cela on définit les structures suivantes :

```
typedef struct Cellule {
        int contenu;                    /* entier */
        struct Cellule suivant;         /* pointeur vers la cellule suivante */
} Cellule;

typedef struct Liste {
        int cardinal;           /* nombre d'elements */
        Cellule *premier;       /* pointeur vers la cellule du premier element */
} Liste;
```

Par exemple, la liste $f1$ comportant les entiers [5, 8, 6, 3] sera représentée ainsi en mémoire :



Écrire :

1. une fonction `creerListe` qui ne prend aucun paramètre, et retourne une liste vide.
2. une fonction `estvide` qui prend en entrée une `Liste`, et teste si celle-ci est vide (retourne 1 si c'est le cas, 0 sinon).
3. une fonction `dernier` qui prend en entrée une `Liste`, et retourne un pointeur vers sa dernière cellule.
4. une fonction `ajouter` qui prend en entrée un entier et une `Liste`, et ajoute cet entier à la Liste.
5. une fonction `afficheListe` qui prend en entrée une `Liste`, ne retourne rien mais affiche les éléments d'une liste.
6. une fonction `contient` qui prend en entrée une `Liste` et un entier, et teste si cet entier appartient à la liste (elle retourne 1 si c'est le cas, et 0 sinon).
7. une fonction `getIth` qui prend en entrée une `Liste` et un entier i, et retourne le i^{eme} élément de la liste (sous forme d'une `Cellule`).