TP: Structures

A chaque fonction, sa fonction de test. On a l'habitude maintenant.

Exercice 1. Pour les tests à réaliser dans ce TP :

1. Ecrire une fonction `void afficheT(int n, int *t)` qui affiche le contenu d'un tableau d'entiers de taille n.
2. Ecrire une fonction `void afficheM(int n, int m, int **t)` qui affiche le contenu d'une matrice d'entiers.

Exercice 2.

1. Déclarer une structure `tab_with_len` contenant un tableau avec 3 champs :
   — Le premier, `next`, est un entier qui indique la première case disponible dans le tableau.
   — Le second, `max`, est un entier qui indique la capacité d'accueil du tableau. On doit donc avoir `.next<.max`
   — Le dernier, `tab` est un pointeur d'entier représentant un tableau.
   L'alias pour représenter cette structure sera noté `twl`.
2. Ecrire la fonction `twl* init(int n)` qui renvoie un pointeur sur un `twl` dont le tableau, de longueur `n`, a toutes ses cases à 0.
3. Écrire une fonction `void afficheTWL(twl *p)` qui prend en paramètre un pointeur sur `twl` et affiche sur deux lignes :
   — la capacité du tableau et la position de la première case disponible.
   — le contenu de ce tableau.
4. Ecrire une fonction `void resize(int n, twl *p)` qui change la longueur du tableau du `twl` pointé par `p`. La nouvelle taille, `n` doit être plus grande que l'ancienne (une erreur d'assertion sera soulevée sinon). Le début du nouveau tableau est une copie de l'ancien.
5. Écrire une fonction `void add(twl *p, int x)` qui met la valeur `x` à la première case libre du tableau et effectue un redimensionnement si nécessaire.

Exercice 3. Dans cet exercice, on considère la structure **date** suivante :

```
typedef struct
{
int jour;
int mois;
int annee;
}date;

```

1. Ecrire la fonction **int nbJours(int mois)** qui retourne le nombre de jours dans un mois donné (1 pour janvier, 2 pour février...) d'une année non bissextile.
   Une erreur d'assertion est soulevée en cas de numéro de mois non valide.
2. Ecrire la procédure **void afficheDate(date d)** qui affiche une date au format jour/mois/année.
3. Ecrire la fonction **bool bissextile(int annee)** qui renvoie **true** si l'année passée en argument est bissextile.
   On rappelle qu'une année est bissextile
   — si l'année est divisible par 4 et non divisible par 100 ;
   — ou bien si l'année est divisible par 400.
   Proposer une batterie de tests exhaustive (ne pas se contenter de prendre une année bissextile et une autre qui ne l'est pas). Il s'agit donc de donner des entrées qui réalisent toutes les possibilités de satisfaire (ou de ne pas satisfaire) le test de bissextilité (faire une table de vérité).
   Remarque. Cet exercice (réaliser une batterie de tests explorant toutes les possibilités de satisfaire une instruction conditionnelle) est très important.
4. Ecrire la procédure **void incDate(date* d)** qui incrémente une date d'un jour avec effets de bord.
5. Ecrire la fonction **date incrDate(date d)** qui retourne la date suivant celle entrée en argument.
6. Ecrire la fonction **bool lt (date d1, date d2)** qui renvoie **true** si la première date est strictement antérieure à la seconde.
7. Ecrire la procédure **void toutes(date d1, date d2)** qui affiche toutes les dates entre le minimum de **d1** et **d2** et le maximum extrêmités incluses.

Exercice 4.

1. Déclaration, initialisation, libération :
   (a) Définissez une structure `matrice` permettant la représentation d'une matrice de valeurs de type double. Cette structure doit contenir le nombre de ligne de la matrice `nligne`, son nombre de colonnes `ncol`, et un champ `mat` (un pointeur).
   (b) Réalisez une fonction **matrice creeMatrice(int n, int m)** qui crée une nouvelle matrice de n lignes et m colonnes avec tous ses éléments initialisés à zéro.
   (c) Ecrire la fonction **void freeMatrice(matrice M)** qui désalloue le pointeur `mat` de la matrice **M**.
2. Réalisez une fonction **void afficheMatrice(matrice* M)** qui affiche à l'écran la matrice pointée par **M**.
   Dans l'hypothèse ou aucun espace n'a encore été alloué au champ du tableau, un message `"matrice vide"` est affiché.
3. Réalisez une fonction **matrice mult(matrice M1, matrice M2)** qui crée une nouvelle matrice représentant le produit des deux matrices **M1** et **M2** (on suppose leurs dimensions compatibles).

Exemple de produit matriciel :

$$\begin{bmatrix} 1 & 2 & 3 \\ -1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} 2 & -4 \\ 1 & 1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ -3 & 7 \end{bmatrix}$$

Exercice 5.

1. Donner une structure **cpx** qui implémente la notion de vecteur de complexe (une partie réelle et une imaginaire).
2. Ecrire la fonction **cpx add (cpx c1, cpx c2)** qui retourne la somme de deux complexes.
3. Ecrire la fonction **cpx mult (cpx c1, cpx c2)** qui retourne le produit de deux complexes.
4. Ecrire la fonction **cpx conjugue (cpx c1)** qui retourne le conjugué d'un complexe.
5. Ecrire la fonction **double module (cpx c1)** qui retourne le module d'un complexe.
6. Ecrire la procédure **void tri_insertion (int n , cpx tab[n])** qui trie par ordre décroissant de modules les éléments d'un tableau de complexes en respectant le tri par insertion.
7. Dans le *tri à bulle* par ordre croissant d'un tableau, l'algorithme parcourt le tableau de la gauche vers la droite et compare les éléments consécutifs. Lorsque deux éléments consécutifs ne sont pas dans l'ordre, ils sont échangés.

   A la fin d'une itération, l'élément le plus grand se trouve donc en fin de tableau.

   Cette opération est répétée jusqu'à ce qu'un parcours ne produise plus aucun échange.

   (a) Ecrire la procédure **void tri_bulle (int n , cpx tab[n])** qui trie par ordre croissant de modules les éléments d'un tableau de complexes en respectant le tri à bulle.

   (b) Analyser la complexité de votre algorithme (meilleur et pire cas).