Séance 11&12 - Prise en main des structures

A FAIRE SUR PAPIER

Exercice 1 Corriger les codes suivants pour qu'ils compilent sans erreur :

```
struct point{
  float x;
  float y;
}

struct segment{
   point a;
   point b;
};

struct cercle {
  point centre;
  float rayon;
};
```

```
void affiche(cercle c){
  std::cout<<"centre="<<c.centre;
  std::cout<<" rayon="<<c.rayon
           <<std::endl;
}
int main(){
  cercle a,b,c;
  a.centre ={1.2,2.4};
  a.rayon = 6.5;
  b= {{6.6,8.2},5}; c=b;
  affiche(&a);
  affiche(&b);
  affiche(&c);
}
```

Exercice 2 Quelles sont les définitions correctes parmi les définitions suivantes ? Corriger celles qui ne compilent pas.

```
  struct poupeeRusse1 {
    float prix;
    char* reference;
  }

 struct poupeeRusse2{
    int reference;
    float prix;
  };
```

```
  struct poupeeRusse3 {
    int reference;
    float prix;
    poupeeRusse3 contenu;
  };

  struct poupeeRusse4{
    int reference;
    float prix;
    poupeeRusse4 * contenu;
  };
```

Corriger les déclarations qui sont fausses.

```
  poupeeRusse1 x;
  poupeeRusse2 y;
  poupeeRusse3[3];
  poupeeRusse2 *t;
  poupeeRusse4* [10];
  poupeeRusse2 [][10];
```

Exercice 3 Pour représenter un nombre complexe, définir un type de structure qui contient deux flottants, la partie réelle et la partie imaginaire.

1. Écrire une fonction `saisie` permettant de saisir un complexe et de le renvoyer.
2. Écrire une fonction `affiche` permettant d'afficher un complexe passé en paramètre.
3. Écrire une fonction `somme` qui reçoit en paramètre deux complexes, qui renvoie la somme des deux complexes.
4. Écrire un programme qui saisit deux complexes c1 et c2 puis ajoute c2 à c1 en mémorisant le résultat dans c1 et enfin affiche c1.

Exercice 4 Écrire une structure `temps` qui comporte trois champs entiers : heure, minute et seconde.

1. Écrire une fonction `conversion` qui prend un nombre réel représentant des secondes et qui renvoie la structure `temps` équivalente (les minutes sont comprises entre 0 et 59, les secondes sont comprises entre 0 et 59. Les heures ne seront pas bornées.).
2. Écrire une fonction `saisie` permettant de saisir les valeurs des champs d'une structure `temps` passée en paramètre. On saisit les données sous la forme hh :mm :ss. la fonction `saisie` contrôle le format des données saisies et redemande une saisie si le format des données n'est pas satisfait.

3. Écrire une fonction `addTime` qui additionne deux structures `temps`. La fonction `addTime` a 3 paramètres, les deux structures à additionner et la structure `temps` résultat de l'addition.
4. Écrire une fonction `compareTime` qui compare deux structures `temps T1` et `T2`, qui renvoie -1 si `T1<T2`, 1 si `T1>T2`, 0 si `T1=T2`.
5. Écrire une fonction `afficheTemps` qui affiche les valeurs des champs de la structure `temps` dont on passe l'adresse en paramètre en respectant le format hh :mm :ss.

A FAIRE SUR MACHINE

Exercice 5 On cherche maintenant à définir une structure `CD` contenant : un nombre de pistes et leurs durées respectives.

1. Écrire la structure `CD`.
2. Écrire une fonction `cree` qui crée un `CD` et le renvoie.
3. Écrire une fonction `init` permettant de saisir les valeurs des champs du `CD` passé en paramètre.
4. Écrire une fonction `afficheCD` qui affiche le nombre de pistes et la suite des durées de chaque plage d'un `CD` dont on passe l'adresse en paramètre.
5. Écrire une fonction `dureeTotale` qui calcule la durée totale d'un CD.
6. Écrire une fonction `nbPistesSup` qui calcule le nombre de pistes dont la durée est supérieure à un nombre de secondes donné en paramètre.
7. Écrire un programme qui crée un CD contenant 3 plages de durées 0 :2 :35, 0 :5 :25, 0 :6 :58, qui affichera la durée totale de ce CD et le nombre de pistes dont la durée est supérieure à 3 minutes.

Exercice 6 Soit le type de structure suivant :

```
1 struct s_point {
2         char nom;
3         int x,y;
4 };
```

1. Écrire une fonction `saisie` qui renvoie un point dont le nom et les coordonnées sont saisies par l'utilisateur. La fonction contrôle si le nom du point est une lettre comprise entre 'A' et 'Z'.
2. Écrire une fonction `affiche` qui reçoit en argument une telle structure et qui en affiche le contenu sous la forme : **point B de coordonnées 10 12**.
3. Écrire une fonction `maz` qui met à zéro les coordonnées d'une telle structure et le nomme '0'.
4. Écrire une fonction qui calcule et renvoie le point de coordonnées opposées et dont le nom sera la lettre symétrique du nom du point par rapport à la fin de l'alphabet.

Exercice 7 En utilisant la structure précédente,

1. Définir une structure `LignePolygonale` contenant un tableau dynamique de points, ainsi que le nombre de points.
2. Écrire une fonction `saisieLP` qui renvoie une ligne polygonale dont l'utilisateur saisit le nombre de points, le nom et les coordonnées de chacun de ses points.
3. Écrire une fonction afficheLP qui affiche chacun des points de la ligne polygonale passée en paramètre.
4. Écrire une fonction `distance` qui calcule la distance de deux points passés en paramètre.
5. Écrire une fonction `longueur` qui calcule la longueur d'une telle ligne polygonale.
6. Écrire une fonction `carre` qui retourne la ligne polygonale de côté n et issue d'un point A tous deux passés en paramètre.
7. Tester votre programme en calculant la longueur de votre carré.

A FAIRE SUR PAPIER

Exercice 8 On souhaite représenter un polynôme par une structure.

1. Écrire une fonction `cree` demandant à l'utilisateur le degré du polynôme et qui renvoie une polynôme du degrés désiré dont les coefficients sont tous à 0.
2. Écrire une fonction `saisieCoeff` permettant de saisir tous les coefficients d'un polynôme passé en paramètre.
3. Écrire une fonction `affiche` permettant d'afficher un polynôme.
4. Écrire une fonction `derivee` calculant la représentation sous forme de polynôme de la dérivée d'un polynôme.
5. Écrire une fonction `primitive` calculant la représentation sous forme de polynôme de la primitive d'un polynôme.
6. Écrire une fonction `evalue` qui évalue un polynôme en un point passé en paramètre.

Exercice 9 On souhaite représenter un vecteur par une structure.

1. Proposez une structure permettant de contenir la taille du vecteur, son nom (sur une seule lettre) ainsi que l'ensemble de ses valeurs.
2. Ecrire une fonction qui crée et initialise un vecteur en mettant sas valeurs à 0. Cette fonction aura comme paramêtre la taille du vecteur ainsi que son nom.
3. Ecrire la fonction qui, à partir d'un vecteur, renvoie le vecteur opposé.
4. Ecrire une fonction qui calcule la somme des vecteurs a et b passés en paramètres (après avoir vérifié qu'ils ont bien la même taille)
5. Ecrire une fonction qui calcule le produit scalaire de deux vecteurs.
6. Utiliser la fonction précédente pour écrire une fonction qui calcule la norme d'un vecteur.

Exercice 10 On voudrait créer une structure de donnée de type pile. Pour cela on définit les structures suivantes :

```
1 struct cellule{
2   int contenu; /* entier */
3   cellule *suivant; /* pointeur vers la cellule suivante */
4 };
5
6 struct pile{
7   int cardinal; /* nombre d'elements */
8   cellule *premier; /* pointeur vers la cellule du premier element */
9 };
```

1. Écrire une fonction qui crée une pile vide ;
2. Écrire une fonction qui renvoie le cardinal d'une pile ;
3. Écrire une fonction `estvide()` qui teste si une pile est vide ;
4. Écrire une fonction `empiler()` qui permet d'empiler un entier sur une pile existante ;
5. Écrire une fonction `depiler()` qui supprime le dernier entier empilé sur la pile et rend sa valeur ;
6. Écrire une fonction qui libère l'espace mémoire occupé par une pile ;
7. Écrire une fonction qui affiche les éléments d'une pile du dernier au premier et une qui les affiche dans l'ordre d'empilement.