TP n°4 – Pointeurs et structures en C

1 Expérimentons avec les pointeurs

Le but de cette section est de créer et manipuler des pointeurs.

- **Q1.** Créer une variable entière x de valeur 14. Créer un pointeur p vers x.
- **Q2.** Afficher l'adresse de x et l'adresse de p. (ce sont des entiers)
- **Q3.** En utilisant `malloc`, créer un pointeur q vers un entier non initialisé.
- **Q4.** Afficher la valeur actuelle de la case vers laquelle q pointe. La réponse peut à priori être n'importe quelle valeur.
- **Q5.** Changer la valeur de *q pour 42.
- **Q6.** Libérer l'espace alloué à q avec `free`.

La mémoire allouée par malloc est non-initialisée. Une variable qu'on a défini uniquement par son type et son nom aussi.

2 Exercices sur les structures

2.1 Les complexes (rappel)

- **Q7.** Définir une structure `struct complexe` pour représenter les complexes.
- **Q8.** En utilisant `typedef`, définir l'alias `complexe` pour `struct complexe`.
- **Q9.** Initialiser l'élément $z = 1,3 + 0,7i$.
- **Q10.** Écrire une fonction `complexe conjugue(complexe z)` qui renvoie le conjugué d'un complexe.
- **Q11.** Écrire une fonction `complexe addition(complexe z1, complexe z2)` qui additionne deux complexes.
- **Q12.** Écrire une fonction `complexe multiplication(complexe z1, complexe z2)` qui multiplie deux complexes.
- **Q13.** Écrire une fonction `conjugue_en_place` qui transforme un complexe en son conjugué par effet de bord (cela signifie que son type de retour est `void`). Que doit prendre en entrée cette fonction pour pouvoir modifier un complexe ?

2.2 Tableau amélioré

On a vu en cours qu'on ne peut pas retrouver la taille d'un tableau et donc qu'il faut toujours la passer en argument des fonctions, avec le tableau.

Pour corriger cela on va créer une structure qui associe un tableau de flottants et sa taille :

```
struct tableau {
 float* contenu;
 int taille;
}

typedef struct tableau tableau;
```

- **Q14.** Écrire une fonction `tableau cree_tableau(float* tab, int n)` qui définit une variable du type `tableau` à partir d'un tableau de flottants de C.
- **Q15.** Écrire une fonction qui prend en entrée un tableau de type `tableau` et un entier i et renvoie l'élément qui se situe à la case i du tableau.
  La fonction doit vérifier que l'indice est valide, on pourra utiliser `assert`.
- **Q16.** Écrire une fonction `void modifie_case(tableau t, int i, float val)` qui change la valeur à la case i du tableau pour y mettre `val`.
- **Q17.** Écrire une fonction qui libère l'espace pris par un tableau de type `tableau`.

2.3 Une liste ?

En Python la structure de stockage de plus commune s'appelle une "liste" (c'est un usage impropre de ce mot, à ne pas retenir). Comme un tableau, elle permet un accès immédiat à toutes ses cases, mais contrairement à un tableau, sa taille est variable et on peut lui ajouter des éléments à la fin.

Ainsi en Python on peut partir de la liste `[0,1,2,3,4]` et utiliser une instruction nommée `append(5)` pour obtenir `[0,1,2,3,4,5]`.

Ce genre de structure n'existe pas naturellement en C. On va donc essayer de l'implémenter, d'un manière naïve, on reviendra sur des méthodes plus efficaces dans le cours sur les listes.

On va se rattacher à ce qu'on connait : les tableaux. Problème : ils ne sont pas de taille modifiable. Une astuce (limitée) est alors de prévoir un tableau avec une grande taille N (qu'on espère assez grande), et de remplir ses cases au fur et à mesure.

Pour savoir où on en est, on stocke l'indice du dernier élément stocké.

Par exemple, imaginons $N = 10$ et la situation suivante (qui représente la liste $[0, 1, 2, 3, 4]$) :

0	1	2	3	4	?	?	?	?	?

L'indice du dernier élément stocké est 4, naturellement un nouvel élément ira à l'indice 5. Les valeurs situées après 4 ne sont pas importantes, elles ne font pas partie de ce qu'on veut stocker.

- **Q18.** Écrire une structure correspondante. N sera un de ses champs.
- **Q19.** Écrire une fonction qui crée une "liste" vide selon notre représentation.
- **Q20.** Pour aider au déboggage, écrire une fonction qui affiche le contenu d'une "liste" à l'écran.
- **Q21.** Écrire une fonction qui calcule et renvoie la taille d'une "liste".
- **Q22.** Écrire une fonction qui ajoute un élément, à la manière de `append` en Python.
- **Q23.** Écrire une fonction qui accède à l'élément i de la "liste".
- **Q24.** Écrire une fonction qui retire le dernier élément de la "liste".