Initiation à la Programmation en C (L1 CPEI)*

TP 9 : Structures et listes chaînées

10/04/2019

Objectifs

Utiliser les structures pour définir et manipuler des listes chaînées.

§1. Rappels sur l'utilisations des structures

Définition. En C, une structure permet de créer des types complexes qui ont plusieurs attributs de type simple (comme des entiers), ou complexe (comme une autre structure).

Pour définir une nouvelle structure, on utilise le mot-clé `struct`. Par exemple, pour définir un point dans l'espace à deux dimensions:

```
1 struct point {
2     float x;
3     float y;
4 };
```

On peut aussi lui donner un alias avec le mot-clé `typedef`. Ici, on donne au type `struct point` le nom `point` :

```
1 typedef struct point point;
```

Déclaration. Pour déclarer une variable du type `point` défini ci-dessus, on utilise la syntaxe suivante :

```
1 point p;
```

On peut aussi donner à `p` des valeurs initiales :

```
1 point p = {1.0, 3.0};
```

*Cours donné par prof. Roberto Amadio. Moniteur 2019 : Cédric Ho Thanh. TPs/TDs basés sur ceux des précédents moniteurs : Florien Bourse (2017), Antoine Dallon (2018). Autres contributeurs : Juliusz Chroboczek, Gabriel Radanne.

Lecture et affectation. Pour accéder à un attribut d'une structure, on utilise l'opérateur `.`, mais si l'on ne dispose que d'un pointeur vers la structure, on doit utiliser l'opérateur `->`. Par exemple :

```
point p = {1.0, 3.0};
point* ptr = &p;
printf("Coordonnees: (%f, %f)", p.x, p.y);
printf("Coordonnees: (%f, %f)", ptr->x, ptr->y); // Équivalent
```

§2. Listes chaînées

Une liste est un ensemble d'éléments dont la taille n'est pas fixée *à priori* (contrairement aux tableaux). Ce type de données est défini récursivement : un liste peut être

- la liste vide ;
- une liste non-vide, composée d'un premier élément appelé *tête*, et d'une liste qui contient la suite des éléments appelée *queue*.
  En C, on définira le type "liste d'entiers" comme suit :

```
typedef struct list list;
struct list {
    int head;
    list* queue;
};
```

Pour passer des listes en argument à des fonctions, un utilisera le type pointeur `list*`. La liste vide est représentée par le pointeur vers `NULL` :

```
list* vide = NULL;
```

À noter : Dans le fichier `tp9.c` fourni sur Moodle, vous trouverez la fonction `list* new_list(int array[], int len)`, qui prend en argument un tableau d'entier ainsi que sa taille, et le converti en une liste chaînée. Vous pouvez vous en servir pour tester vos solutions.

Exercice 1 (Affichage)**.** (1) Écrivez une fonction qui prend en argument une liste, et qui affiche tous les entiers présents dans la liste, dans l'ordre.
(2) Écrivez une fonction qui prend en argument une liste, et qui renvoie la taille de la liste.
(3) Écrivez une fonction qui prend en argument une liste `lst`, et qui renvoie une liste qui contient les mêmes entiers que `lst` mais dans l'ordre inverse.

Exercice 2 (Recherche)**.** (1) Écrivez une fonction qui prend en argument une liste et un indice i, et qui renvoie l'entier en position i dans cette liste. Si cet entier n'existe pas, cette fonction devra renvoyer la valeur -1.

(2) Écrivez une fonction qui prend en argument une liste et un entier x et qui renvoie l'indice de la première occurrence de x dans la liste. La fonction renverra la valeur -1 si la liste ne contient pas x.

(3) Écrivez une fonction qui prend en argument une liste et qui renvoie l'indice de la valeur minimale de la liste. La fonction renverra `INT_MIN` si la liste est vide. Pour utiliser la constante `INT_MIN`, il faut inclure la librairie `limits.h`:

```
1 #include <limits.h>
```

Exercice 3 (Tri)**.** • Écrivez une fonction qui prend en argument une liste et deux entiers i et j, et qui échange les entiers en position i et j.

• Écrivez une fonction qui prend en argument une liste et la trie.