

TD10 – Pointeurs sur fonctions

Langage C (LC4)

semaine du 5 au 11 avril 2010

1 Utilisation simple

Question 1. Écrire une fonction `void doubler(int n, int * tableau)` qui multiplie par 2 chacun des `n` éléments du tableau.

Question 2. Écrire une fonction `void appliquer_tableau(int (*f)(int), int n, int *tableau)` qui applique une fonction `f` à chacun des `n` éléments du tableau.

Question 3. Réécrire `doubler` en utilisant `appliquer_tableau`. On pourra s'aider d'une fonction `int fois_deux(int i)` qui renvoie le double de l'entier qui lui est donné en argument.

2 Itérations dans un tableau par programmation fonctionnelle

2.1 Extréma d'un tableau

Question 4. Écrire une fonction `int minimum(int n, int * tableau)` qui calcule le plus petit élément du `tableau` supposé de taille `n`. Si `n` vaut zéro (tableau vide), alors on renverra la constante `INT_MAX`¹

Question 5. Que faut-il changer pour calculer le plus grand élément du tableau (et renvoyer `INT_MIN` si le tableau est vide) ?

Question 6. Écrire une fonction `extremum` qui calcule l'extrémum d'un tableau suivant un ordre et une valeur par défaut donnés en argument. On supposera que l'ordre \triangleleft donné en argument est une fonction qui prend deux entiers a et b en argument et renvoie une valeur non nulle si et seulement si $a \triangleleft b$.

Question 7. Réécrire les fonctions `minimum` et `maximum` en utilisant `extremum`. On pourra définir des fonctions auxiliaires `ordre_minimum`, `ordre_maximum`.

2.2 Somme des éléments d'un tableau

Question 8. Écrire une fonction `int somme(int n, int * tableau)` qui calcule la somme des éléments du `tableau` supposé de taille `n`. Si le tableau est vide, alors on renverra 0.

¹Cette constante ainsi que `INT_MIN` sont définies dans `limits.h`

Question 9. Sur ce modèle, écrire une fonction `iterations` qui, étant donnés un opérateur \top , une valeur initiale Z et un tableau t , calcule $((Z \top t_0) \top t_1) \dots \top t_{n-2}) \top t_{n-1}$.

Question 10. Réécrire la fonction `somme` en utilisant `iterations`.

2.3 Généralisation

Question 11. Pourquoi n'est-il pas possible de réécrire la fonction `extremum` en utilisant directement `iterations` ?

Question 12. Pour pallier cet inconvénient, nous allons supposer que l'opérateur \top donné en argument de `iterations` prend un argument supplémentaire de type `void *`, et que cet argument supplémentaire est aussi fourni en argument de `iterations` qui se contentera de le passer à l'opérateur \top .²

Modifier la fonction `iterations` dans ce sens.

Puis, grâce à ces modifications, réécrire la fonction `extremum` en utilisant directement la nouvelle version de `iterations`, avec l'argument supplémentaire bien choisi. Puis appliquer en réécrivant les fonctions `minimum` et `maximum`.

On rappelle que `void *` est un type permettant de manipuler un pointeur vers des données quelconques : pour tout type T , une donnée de type T^* est de type `void *`.

2.4 Comptages

Question 13. Écrire, en utilisant `iterations`, une fonction `pairs` qui permet de compter dans un tableau le nombre d'éléments pairs.

Question 14. Généraliser en écrivant une fonction `compte` qui, en utilisant `iterations`, permet de compter le nombre d'éléments vérifiant un certain prédicat P . On supposera que P est donné sous la forme d'une fonction prenant un argument entier a et renvoyant une valeur non nulle si et seulement si $P(a)$ est vraie.

Question 15. Que faudrait-il faire si on voulait passer par les étapes intermédiaires :

- comptage des éléments divisibles par un entier d donné en paramètre
- comptage des éléments plus petits qu'un entier d au sens d'un ordre \triangleleft donné en paramètre (la divisibilité pouvant être vue comme un cas particulier)

²Cet argument est appelé *fermeture* (ou *closure* en anglais).