TP 3

Juliusz Chroboczek et Gabriel Scherer

31 octobre 2014

Exercice 1 (Préliminaires)**.**

1. Écrivez une fonction `void print_array(int a[], int len)` qui prend un tableau d'entiers (et sa taille) et affiche tous ses éléments, séparés par des espaces, puis va à la ligne.
2. Écrivez une fonction `void read_array(int a[], int len)` qui prend un tableau (et sa taille) et le remplit avec autant d'entiers demandés à l'utilisateur.

Écrivez une fonction `main` pour tester ces deux fonctions. Gardez-les sous la main ; elles seront utiles pour tester les fonctions que l'on vous demande — ce qui est nécessaire même quand ce n'est pas précisé.

Exercice 2.

1. Écrivez une fonction `int max(int a[], int len)` qui renvoie le plus grand élément du tableau d'entiers fourni.
2. Écrivez une fonction `int max_pos(int a[], int len)` qui renvoie *l'indice* du plus grand élément du tableau d'entiers fourni.
3. Écrivez une fonction `int search(int a[], int len, int v)` qui renvoie la plus petite position `i` telle que `a[i] == v`, ou bien `-1` si tous les éléments du tableau sont différents de `v`.

Exercice 3. `#include <math.h>`

1. Écrivez une fonction `double phi(double x, int n)` qui à x et n associe $\left(\frac{1+x\sqrt{5}}{2}\right)^n$.

   Vous pourrez utiliser la fonction `double pow(double x, double y)` pour le calcul de la puissance x^y, et `sqrt` pour la racine carrée [1].
2. Écrivez la fonction `double fun(int n)` qui à n associe $\frac{(1+\sqrt{5})^n+(1-\sqrt{5})^n}{2}$.
3. Écrivez un programme qui affiche les valeurs de cette fonction `fun` pour n allant de 1 à 15, séparées par des espaces. Le résultat vous rappelle-t-il quelque chose ?

Exercice 4.

1. Écrivez une fonction swap `int swap(int i, int j, int a[], int len)` qui échange les valeurs aux indices `i` et `j` du tableau `a`.

1. Il faudra ajouter quelque chose à la ligne de commande du compilateur — `man 3 pow` est votre ami.

2. Remarquez qu'on peut utiliser la fonction `max_pos` de l'exercice 2 en donnant une taille plus petite que la taille réelle du tableau. Écrivez un programme qui lit un tableau de taille `10` en entrée, échange son plus grand élément avec son dernier élément, puis son deuxième plus grand élément avec son avant-dernier élément — et affiche le résultat.
3. Écrivez une fonction `int selection_sort(int a[], int len)` qui trie un tableau en répétant l'opération de la question précédente.

Exercice 5. `#include <stdbool.h>`

1. Écrivez une fonction `bool prime(int n)` qui renvoie `true` si n est premier ; un nombre n est premier s'il est supérieur ou égal à 2 et n'a aucun diviseur autre que 1 et lui-même.
2. Écrivez un programme qui lit un entier `n` et affiche le nombre de nombre premiers entre 2 et n. Par exemple, entre 2 et 10 il y a exactement 4 nombres premiers $(2,3,5,7)$.

On va maintenant utiliser une technique efficace inventée par le programmeur grec Eratosthène pour trouver tous les entiers premiers entre 2 et n : on crée un tableau de booléens de taille $n+1$, dont l'on met toutes les cases à `false` (ce nombre n'a pas encore été parcouru par l'algorithme). Ensuite, pour tout nombre i entre 2 et n :

— si la case i du tableau est à `true`, on a déjà parcouru cette case donc i n'est pas premier

— sinon, i est premier ; on parcours les cases de tous les multiples de i inférieurs à n pour les mettre à `true` — ils ne sont pas premiers

Utilisez cette méthode pour compter tous les nombres premiers entre 2 et n, et comparez les performances avec la méthode précédente (on pourra utiliser `time(NULL)` pour mesurer le temps d'exécution).

Exercice 6 (Un peu d'aléa)**.** On dit qu'un générateur aléatoire est *uniforme* si tous les résultats ont la même chance de se produire — les probabilités sont égales. Un dé uniforme ne renvoie aucun nombre plus souvent que les autres.

(Les deux questions qui suivent sont indépendantes.)

1. Écrivez une fonction `int flip()` qui renvoie soit 0 soit 1, de façon uniforme. En utilisant seulement la fonction `flip()` (aucune autre génération de hasard), écrivez une fonction qui renvoie aléatoirement 0, 1 ou 2, de façon uniforme.
2. Écrivez une fonction `void shuffle(int a[], int n)` qui mélange aléatoirement les éléments d'un tableau. Cette permutation est-elle uniforme — tous les réordonnement ont-ils autant de chance d'être choisis ?