TP 3 Fonctions

12 octobre 2017

1 Fonctions pures

1. Écrire des fonctions de signature :

   ```
   double perimeter(double radius)
   double area(double radius)
   ```

   qui renvoient respectivement le périmètre et l'aire d'un cercle de diamètre `radius` donné en argument.
2. Modifiez le code que vous avez écrit pour la question 1.3. du TP 1, de façon à utiliser les deux fonctions de la question précédente.
3. Ecrire une fonction qui calcule et renvoit la somme des n premiers carrés :

   $$1 + 2^2 + 3^2 \cdots + n^2$$

   L'entier n est un argument de la fonction.
4. Écrire des fonctions de signature :

   ```
   double min(double[] tab)
   double max(double[] tab)
   ```

   qui renvoient respectivement le plus petit et le plus grand des éléments du tableau `tab` passé en argument. Testez-les dans la fonction `main` en affichant leur retour sur des exemples.
5. Écrire une fonction de signature :

   ```
   bool equals(double[] t1, double[] t2)
   ```

   qui renvoit `true` si `t1` et `t2` sont égaux (contiennent à chaque indice des éléments égaux).
6. À l'aide des fonctions précédentes, écrire une fonction :

   ```
   double[] normalize1(double[] tab)
   ```

   qui renvoit un tableau de `doubles` entre 0 et 1, et ayant les même rapports de proportionnalité entre ses éléments que le tableau argument `tab`. Autrement dit, si m est le minimum de `tab` et M son maximum, chaque élément d'indice i du tableau retourné vaudra

   $$\frac{\texttt{tab[i]} - m}{M - m}$$

   Attention : la fonction ne devra pas modifier le tableau `tab` mais réallouer un nouveau tableau et le renvoyer. [1]

1. i.e., elle traitera `tab` comme étant *persistent*.

2 Procédures

1. Écrire une fonction de signature :

   `void normalize2(double[] tab)`

   qui fait la même chose qu'en 1.5., mais en modifiant directement le tableau argument `tab`. [2]
2. Écrire une fonction de signature :

   `void printDoubleArray(double[] tab)`

   qui affiche le contenu d'un tableau `tab` de flottants double précision.
3. Testez la différence entre traitement *persistent* et traitement *en place* d'un tableau. Exécutez d'abord ce programme :

```
double[] t = {1.32, 2.6, 3.333, 0.05, 7.0}
printDoubleArray(t);
double[] t2 = normalize1(t);
printDoubleArray(t);
printDoubleArray(t2);
```

   Modifiez-le ensuite pour utiliser `normalize2`. Qu'observez-vous ?

3 Récursivité

1. Écrire la fonction factorielle de façon récursive : elle prendra un entier n en argument, et renverra $n!$. On rappelle la définition de la factorielle :

$$0! = 1 \qquad\qquad n! = n \times (n-1)!$$

2. On va maintenant programmer l'algorithme de *recherche dichotomique*. Imaginons que l'on veuille rechercher si un entier `n` est présent dans un tableau `t`. On doit potentiellement parcourir tous les éléments du tableau jusqu'à la fin. Imaginons maintenant que le tableau `t` a tous ses éléments triés par ordre croissant (pour tout indice $\texttt{i} \geq 1$, $\texttt{t[i]} > \texttt{t[i-1]}$) ; on peut alors écrire un algorithme plus rapide. On part de la case du milieu du tableau ; appelons `p` son contenu. Si par hasard $\texttt{p} = \texttt{n}$, alors on a trouvé notre élément et on renvoit `true`. Si $\texttt{p} < \texttt{n}$ on sait que l'on doit maintenant regarder dans la moitié supérieure du tableau, car la moitié inférieure ne contiendra que des nombres inférieurs à `p` ; sinon si $\texttt{p} > \texttt{n}$, on doit maintenant regarder dans la moitié inférieure du tableau. On se focalise donc sur la moitié qui nous intéresse, et on recommence le processus : on regarde la case du milieu de notre nouvelle moitié, on choisit une moitié de moitié etc. jusqu'à ce que le sous-tableau à traiter soit vide.
   - Écrire la fonction auxiliaire :
     `bool dicho(int n, int[] t, int i, int j)`
     qui prend un entier `n` et un tableau *supposé donné trié* `t` et deux bornes `i` et `j`, et fait la recherche dichotomique de `n` dans le sous-tableau composé des cases adjacentes `t[i]`, `t[i+1]`...`t[j]`.
   - Écrire finalement la fonction principale :
     `bool recherche_dicho(int n, int[] t)`
     qui prend un entier `n` et un tableau *supposé donné trié* `t`, et renvoit `true` si `n` est présent dans `t` et `false` sinon. Elle utilisera évidemment la fonction `dicho` définie plus haut.

2. i.e. elle traitera `tab` *en place*.