PHP

Exercice 1 : *Tris selon un ordre choisi par le développeur*

Les fonctions `sort()`, `asort()`, `ksort()`, vues en cours, ne peuvent être utilisées que pour trier selon l'ordre prédéfini (ordre naturel sur les nombres, ordre lexicographique sur les chaînes, etc) .
Pour trier des valeurs selon un ordre « personnalisé », il va falloir définir d'abord une fonction de comparaison, et utiliser des fonctions de tris qui en tient compte.

Une **fonction de comparaison** est une fonction à écrire par le développeur : elle prend en paramètre 2 valeurs à comparer. Le résultat indique quelle est la « plus petite » selon l'ordre que l'on souhaite implémenter. Une fonction de comparaison (appellons la *fcomp*) devra vérifier :

fcomp`($x,$y)` < 0	si `$x` est à considérer comme `< $y`
fcomp`($x,$y)` > 0	si `$x` est à considérer comme `> $y`
fcomp`($x,$y)` $== 0$	si `$x == $y` ou si `$x` et `$y sont équivalents`

Par exemple, si on veut trier des nombres selon leur valeur absolue, la fonction de comparaison de 2 nombres x et y pourra renvoyer $|x| - |y|$:

```
function compareAbs($x, $y){
    return abs($x)-abs($y);
} //  l'ordre de classement qui en découle est celui des valeurs absolues croissantes.
```

`compareAbs(2, -5) < 0` (2 est considéré comme plus petit que -5)
`compareAbs(-5, -6) < 0` (-5 est considéré comme plus petit que -6)
`compareAbs(-3, 3) == 0` (-3 et 3 sont équivalents)

Pour réaliser le tri d'un tableau « classique » (indexé par des entiers successifs) selon cet ordre il faut utiliser la fonction `usort(...)` (à la place de `sort(...)`).
On passe comme 2nd argument de `usort(...)` le nom de la fonction de comparaison sous forme de chaîne.

```
usort($tableau,"compareAbs")
```

Exemple

```
$ta=[-6,3,-5]; $tb=[-3, 8, -5, 3];
usort($ta,"compareAbs")} // transforme $ta en [3,-5,-6]
usort($tb,"compareAbs")} // transforme $tb en [3,-3,-5,8]  ou [-3,3,-5,8]
```

On notera que notre fonction de comparaison a considéré que 2 nombres opposés étaient équivalents (résultat 0). En cas de tri, plusieurs résultats sont donc possibles. (exemple ci-dessus).
Si l'on voulait assurer que, en cas de valeurs absolues égales, la valeur négative soit considérée comme plus petite on pourrait coder

```
function compareAbs($x, $y){
   $cmp = abs($x)-abs($y);
   if ($cmp != 0)
     return $cmp
   else
    return $x - $y;
}
```

Attention : l'implémentation de la fonction de comparaison devra respecter quelques règles :

- *fcomp*`($x,$x)` vaut 0
- *fcomp*`($x,$y)` et *fcomp*`($y,$x)` sont de signes opposés.
- si *fcomp*`($x,$y)`<0 et si *fcomp*`($y,$z)`<0, alors *fcomp*`($x,$z)`<0 (transitivité)

NB : pour trier des tables associatives, il existe les fonctions `uksort` et `uasort`

Copiez sur le serveur le dossier `tris` (dans l'archive jointe) et son contenu. La structure est analogue à celle de la séance précédente (sous-dossier `lib` ...). Les fonctions figureront dans un fichier du répertoire `lib`. Un script `test.php` vous est fourni.

Q 1 . (illustration) Essayez des tris avec la fonction `compareAbs()`, en utilisant la première partie du script de test.

Faites des essais avec d'autres exemples de tableaux.

Q 2 . (illustration) NB : pour cette question, comme pour les suivantes, on considère que les chaînes ne comportent pas de lettres accentuées.

La fonction prédéfinie `strcmp()` est une fonction de comparaison de chaînes de caractères, selon l'ordre lexicographique (c'est à dire l'ordre usuel sur les chaînes).

Triez le tableau de chaînes fourni selon l'ordre défini par `strcmp()` puis affichez sa valeur.

Constatez qu'on obtient le même résultat si on utilise la fonction `sort()`.

Q 3 . (application directe)

1. Définissez une fonction `comparerChainesParLongueur()` qui compare 2 chaînes. Une chaîne sera considérée comme plus petite si sa longueur est plus petite.

   Définissez un tableau de chaînes, triez-le selon cet ordre et affichez le tableau trié.

2. La fonction de comparaison précédente renvoie 0 quand on l'applique à deux chaînes de même longueur (ce qui était demandé). Quand on l'utilise pour trier, le rangement obtenu entre chaînes de même longueur est donc indéterminé.

   On souhaite assurer que, après tri, les chaînes de même longueur apparaitront par ordre lexicographique **inverse**.

   Définissez pour cela une fonction `comparerChainesParLongueurPlus()` qui convient. Testez l'effet du tri avec cette nouvelle fonction de comparaison.

Exercice 2 :

Vous allez reprendre l'exercice de la feuille précédente, concernant les livres.

Q 1 . Il s'agira cette fois de représenter en mémoire la totalité des ouvrages décrits dans le fichier texte, sans les afficher au fur et à mesure de leur lecture.

1. Dans la feuille précédente, la fonction `libraryToHTML($file)` lisait le contenu du fichier et renvoyait la représentation HTML des livres décrits dans ce fichier. Vous aller maintenant remplacer cette fonction par deux autres :

   (a) `loadBiblio($file)` dont l'argument est un fichier préalablement ouvert. Le résultat est un **tableau PHP** indexé par des entiers successifs. Chaque élément du tableau est un tableau PHP associatif représentant un livre (un tableau renvoyé par la fonction `readBook()`)

   (b) `biblioToHTML($liste)` dont l'argument est un tableau de livres (typiquement : le résultat d'un appel à la fonction `loadBiblio($file)`).

   Le résultat de `biblioToHTML($liste)` est une chaîne contenant la représentation HTML des livres de la liste, dans l'ordre de la liste fournie.

   Attention : comme précédemment, cette fonction ne doit produire aucune entrée/sortie (aucun "echo").

2. Écrivez une nouvelle version du script `bibliotheque.php` qui utilisera ces nouvelles fonctions. Vous l'appellerez `bibliotheque2.php`

Q 2 .

1. Définissez une fonction de comparaison de 2 livres. Cette fonction de comparaison, employée comme critère de tri, doit faire apparaître les livres selon l'ordre lexicographique des titres.

2. Ajoutez à la fonction `biblioToHTML` un argument **optionnel** `$sort`. Si `$sort` vaut `"titles"` alors les livres sont présentés par ordre croissant des titres. Si `$sort` vaut `"none"` (valeur par défaut) les livres sont présentés dans l'ordre de la liste reçue en argument. Dans tous les autres cas une exception est déclenchée.

3. Modifiez le script `bibliotheque2.php` pour qu'il affiche les livres par ordre alphabétique des titres.

Q 3 . *Paramétrage du script*
Vous allez travailler à une nouvelle version du script : commencez par faire une copie de `bibliotheque2.php` sous le nom `bibliotheque3.php`

> *Rappel : paramétrage en mode « GET »*
> *Si le script* `bibliotheque3.php` *est invoqué par le navigateur via une url de la forme*
>
> `......bibliotheque3.php?ordre=xxx`
>
> *alors le script PHP dispose de la variable PHP* `$_GET['ordre']` *qui contient la chaîne* `'xxx'`.
> *S'il est appelé sans l'argument* `ordre`, *alors* `$_GET['ordre']` *est indéfini (ce qui peut être testé avec la fonction* `isset()`*)*

Le script `bibliotheque3.php` aura le comportement :
- appelé avec un argument `ordre` valant `titres`, les livres seront affichés par ordre croissant des titres.
- appelé sans l'argument `ordre` **ou** avec `ordre` valant `aucun`, les livres seront affichés selon l'ordre du fichier de données.
- dans tous les autres cas les livres ne sont pas affichés.

Vous testerez le script dans les différentes situations possibles, en essayant différentes URL, par exemple

```
.....bibliotheque3.php?ordre=titres
.....bibliotheque3.php?ordre=aucun
.....bibliotheque3.php
```

Q 4 . On souhaite maintenant pouvoir afficher les livres par ordre de catégorie et, au sein d'un même catégorie, par années de parution croissantes, et au sein d'une même année par ordre alphabétique des titres.
Cet ordre sera obtenu en appelant le script :

```
.....bibliotheque3.php?ordre=categories
```

(note : cette option de tri s'ajoute aux précédentes)

1. Complétez la bibliothèque de fonctions pour implémenter cette nouvelle relation d'ordre.
2. Adaptez `bibliotheque3.php` pour prendre en compte la nouvelle option.