Feuille de travaux pratiques n° 1
Représentation de l'information

Exercice 1.1

Écrire une fonction `base_converter()` prenant en entrée une chaîne de caractères (`string` C++ ou `char[]` C), une base de départ, une base d'arrivée, et retournant une chaîne de caractères correspondant à la chaîne initiale exprimée dans la base d'arrivée. On considérera que les bases de départ et d'arrivée sont comprises entre 2 et 36 inclus.

Exemple d'utilisation :

```
base_converter("3eh12",18,30) -> "eqne"
```

On prendra soin de gérer les cas d'erreurs.

Exercice 1.2

Combien de caractères comportent les chaînes `"See you soon!"` et `"À bientôt !"` ?

1. Vérifiez votre résultat en écrivant un programme C++ qui stocke ces deux chaînes dans des objets de type `std::string` et qui affiche leurs tailles en utilisant la méthode `.length()`. On s'assurera au préalable que le fichier de code utilise l'encodage UTF-8. Que se passe t-il ? Justifiez ?
2. Écrire le code d'une fonction `strlength()` prenant en entrée un objet de type `std::string` et retournant un compte correct du nombre de caractères qu'elle contient.

Exercice 1.3

Écrire une fonction `std::string retourner(const std::string& s)` retournant une chaîne contenant tous les caractères de `s` dans l'ordre inverse. La chaîne `s` est codée en UTF-8.

Exercice 1.4

On souhaite calculer le plus précisément possible une somme S de nombres flottants en double précision se trouvant dans un tableau `T` de taille n :

$$S = \sum_{i=0}^{n-1} T_i$$

Pour cela, on va comparer expérimentalement la qualité de différents algorithmes de sommation.

1. **Somme récursive.** Il s'agit de l'algorithme le plus simple, où l'on ajoute chacun des T_i dans un accumulateur dans l'ordre où ils apparaissent dans le tableau. Écrire la fonction `somme_recursive()` prenant en paramètre un tableau `T` et retournant la somme de ses éléments (Attention : la fonction est purement itérative et **non récursive**) ;
2. **Somme récursive inverse.** On inverse le tableau `T` et l'on applique une somme récursive sur le nouveau tableau. Écrire la fonction `somme_recursive_inverse()` prenant en paramètre un tableau `T` et retournant la somme de ses éléments dans l'ordre inverse de leurs positions ;
3. **Somme en valeurs (dé)croissantes.** On trie le tableau `T` dans l'ordre (dé)croissant des valeurs, puis l'on applique l'algorithme de somme récursive.
   (a) Écrire la fonction `somme_croissante()` évaluant la somme des éléments du tableau `T` passé en paramètres pour un tri en ordre croissant ;
   (b) Écrire la fonction `somme_decroissante()` évaluant la somme des éléments du tableau `T` passé en paramètres pour un tri en ordre décroissant.

4. **Somme en valeurs absolues (dé)croissantes.** On trie le tableau `T` dans l'ordre (dé)croissant des valeurs absolues, puis l'on applique l'algorithme de somme récursive.
   (a) Écrire la fonction `somme_abs_croissante()` évaluant la somme des éléments du tableau `T` passé en paramètres pour un tri en ordre croissant des valeurs absolues ;
   (b) Écrire la fonction `somme_abs_decroissante()` évaluant la somme des éléments du tableau `T` passé en paramètres pour un tri en ordre décroissant des valeurs absolues.

Tester les fonctions écrites avec le fichiers de données `donnees_somme.h` ou `donnees_somme.hpp` se trouvant sur *madoc*. Justifier les observations faites. Déterminer les points forts et les points faibles de chaque méthode ; vérifier les conclusions à l'aide de jeux de données à définir. Proposer d'autres méthodes de sommation précises.