Travaux Pratiques thème 3 : Les fonctions et les chaînes de caractères

Exercice 1

Écrire un algorithme récursif qui affiche l'écriture inverse d'un entier donné en entrée (par exemple si on entre -2344, on obtiendra l'affichage -4432).

Exercice 2 – Problème « Couicable »

1. Écrire une fonction `nbDeChiffres(x)` qui retourne le nombre de chiffres composant un entier x transmis en arguments.
2. Écrire une fonction `extraitNombre(x,n,lg)` qui extrait de x entier positif, à partir du énième chiffre en partant de la droite, le nombre composé de lg chiffres. Elle renvoie -1 si les entrées ne sont pas correctes.
   Exemple 1, `extraitNombre(45863047,4,3)` renvoie 586.
   Exemple 2, `extraitNombre(45863047,6,4)` renvoie -1.

3. Écrire une fonction `estPair(x)` qui renvoie 1 si le nombre de chiffres de x est pair, 0 sinon.
4. Écrire une fonction `sommeDesChiffres(x)` qui renvoie la somme des chiffres composant le nombre x transmis en arguments
5. Un nombre x est dit *couicable* si la somme des chiffres de sa partie droite est égale à la somme des chiffres de sa partie gauche. Le nombre de chiffres composant le nombre étudié doit être pair. Par exemple, 256823 est couicable.
   Écrire la fonction `estCouicable(x)` qui affiche si x, transmis en arguments, est un nombre couicable ou non.

Exercice 3 - Modification de chaînes

Nous vous rappelons qu'un tableau de caractères peut être initialisé avec une chaîne de caractères. Dans ce cas, en plus d'être utilisable comme n'importe quel autre tableau, il a les caractéristiques des chaînes de caractères : le caractère "\0" est ajouté à la suite du dernier caractère, pour signaler la fin de la chaîne.

Ecrire une fonction `supprime_espaces` qui transforme une chaîne de caractères contenant une phrase (avec espaces) en une chaîne de caractères ne contenant pas d'espace et qui compte le nombre d'espaces supprimés. Tester votre fonction sur différentes chaînes.

Exercice 4 - Miroir

1. Écrivez un programme qui, étant donnée une chaîne de caractères stockée dans un tableau `chaine`, écrit dans le tableau `miroir` le miroir de `chaine` puis l'affiche à l'écran. Le miroir d'une chaîne correspond à la chaîne de caractères obtenue quand on lit la chaîne initiale de droite à gauche. Par exemple, le miroir de "bonjour" est "ruojnob", et le miroir de "rever" est "rever". Pour pouvoir déclarer le tableau `miroir` sans compter le nombre de caractères de `chaine`, vous pouvez l'initialiser avec la même valeur que `chaine`. N'oubliez pas de tester votre programme avec des chaînes de caractères de différentes tailles.
2. Recopiez le programme de la question précédente et modifiez le pour que la chaîne miroir soit écrite dans le tableau `chaine`. Si initialement le tableau `chaine` contient "bonjour", il devra contenir "ruojnob" à la fin du programme. Contrainte : votre programme ne doit pas utiliser d'autre tableau que la variable `chaine`.
3. Écrivez et testez une fonction **récursive** `affiche_miroir` qui prend comme seul paramètre un mot décrit par le tableau `chaine` et qui affiche le mot miroir de `chaine`.

Exercice 5 - Palindrome

Écrivez un programme qui indique si une chaîne de caractères est un palindrome. Un palindrome est un mot qui se lit de la même manière à l'endroit et à l'envers (par exemple les mots “rever” et “radar” sont des palindromes). Vous testerez votre programme avec différentes chaînes de caractères que vous devrez indiquer en commentaire dans le programme en justifiant ce que chaque chaîne choisie vous permet de tester.

Exercice 6 - Sous-chaînes

Écrire une fonction qui prend en entrée deux chaînes de caractères `s1` et `s2` et modifie `s1` de façon à supprimer la sous-chaîne `s2`. Si `s2` n'est pas incluse dans `s1`, la fonction ne modifie pas `s1`. Si `s2` est présente plusieurs fois dans `s1`, on supprime chaque occurence.

Exercice 7 - Parenthèses

Les parenthèses permettent, par exemple, de changer la priorité des opérations dans une expression algébrique. Ex : $(a + 2 * (x - y)) * (b + 2)$. Si les parenthèses permettent de définir une expression algébrique valide, on dit alors que l'expression est bien parenthésée. Dans le cas contraire, l'expression est dite mal parenthésée. Le but de cet exercice est d'écrire un programme permettant de tester la validité des parenthèses dans une expression algébrique.

1. Une expression est considérée comme une chaîne de caractères. Une première étape est de supprimer les caractères de l'expression. Par exemple l'expression donnée précédemment devient : (())(). Écrire une fonction qui prend une expression algébrique en entrée et renvoie un tableau de caractères ne contenant que les parenthèses (ce tableau sera un tableau statique passé en entrée qui sera modifié par la fonction).

2. Dire si les expressions suivantes sont bien ou mal parenthésées. Dans le cas où elles sont mal parenthésées, expliquer pourquoi :
   — ()(())
   — ()))
   — ())(()

3. Proposer une fonction prenant en entrée un tableau de caractères ne contenant que des parenthèses et renvoyant 1 si l'expression est bien parenthésée, 0 si elle est mal parenthésée.

4. Proposer une fonction prenant en entrée un tableau de caractères T ne contenant que des parenthèses et renvoyant la plus longue expression bien parenthésée contenue dans T.