LIFAP1 – TP9 : Début sur les chaînes de caractères

Objectifs : Manipulation des chaines de caractères

1. Fonctions classiques sur les chaînes

a. Définissez en C une constante CHMAX contenant la taille maximale des tableaux de caractères manipulés. Il sera alors possible de stocker des chaînes de longueur CHMAX-1 à cause du '\0' terminal.
b. Ecrivez le programme principal qui permet la saisie et l'affichage d'une chaine de caractères.
c. Écrivez la procédure `chMiroir` qui modifie une chaine de caractères en son miroir. Attention : cette procédure n'affiche pas le résultat à l'écran. Par exemple, "maison" donnera "nosiam".
d. Écrivez la procédure `chConcat` qui concatène deux chaines de caractères (sans utiliser les fonctions de string.h). Le résultat sera stocké dans la première chaine de caractères. Par exemple, la concaténation des deux chaînes "rouge " et "vert" donnera "rouge vert".
e. Écrivez la fonction `chCompare` qui renvoie vrai si deux chaînes de caractères sont identiques, faux sinon (sans utiliser les fonctions de string.h).

Remarque : il sera maintenant possible d'écrire « *Si chCompare(ch1,ch2) alors ... sinon ...* »
Rappel, ceci est interdit : *Si ch1==ch2 alors ...*

f. Écrivez une fonction `menu` qui propose toutes les opérations disponibles à l'utilisateur et retourne un entier correspondant à son choix.
g. Ecrivez le programme principal qui propose à l'utilisateur le menu et appelle les différents sous-programmes.

2. **Deux mots sont des anagrammes** s'ils contiennent exactement les mêmes lettres. Par exemple VILLEURBANNE / INVULNERABLE sont des anagrammes.

a. Écrire une fonction booléenne `TOUT_MIN_OU_TOUT_MAJ` qui prend en paramètre une chaine de caractères et retourne vrai si cette chaine est constituée uniquement de caractères minuscules ou uniquement de caractères majuscules, faux sinon.
b. Écrire une fonction `COMPTE_OCCUR` qui prend en paramètres un caractère et une chaine de caractères et retourne le nombre d'occurrences du caractère dans la chaine.
c. Écrire une fonction booléenne `ANAGRAMME` qui prend en paramètres 2 chaines de caractères et retourne vrai si elles sont anagrammes l'une de l'autre et faux sinon. Pour cela, on commencera par vérifier que leurs longueurs sont identiques puis on vérifiera que le nombre d'occurrences de chaque caractère de la première est identique dans la seconde. On pourra utiliser la fonction *longueur* qui retourne le nombre de caractères d'une chaine passée en paramètre.
d. Écrire le programme principal qui effectue la saisie de deux chaines de caractères tant que celles-ci ne sont pas soit totalement constituées de caractères minuscules soit totalement constituées de caractères majuscules puis teste si les deux chaines sont des anagrammes ou non et affiche le résultat à l'utilisateur.

Pour aller plus loin …

Une autre version des palindromes.

Nous avons conçu en TD un algorithme permettant de dire si un mot est un palindrome. Cet algorithme exploitait le sous-programme `miroir`. Nous allons à présent définir une autre version de ce programme `palindrome` en suivant les étapes suivantes :

a. Écrire l'algorithme d'un sous-programme **`premder`** permettant de constituer une chaîne de caractères en regroupant les caractères de la manière suivante : le premier et le dernier caractère, puis le deuxième et l'avant-dernier, etc.
   Exemple : abcdefg → agbfced
b. On constate que si on applique le sous-programme précédent à un palindrome (exemple : laval → llaav), la chaîne résultat est composée de paires de caractères identiques (sauf le dernier dans le cas impair). Écrire une fonction booléenne **`estpalindrome`** permettant de vérifier si une chaîne passée en paramètre est un palindrome, en utilisant **`premder`**.
c. Écrire le programme principal permettant de saisir une chaîne de caractères, et de vérifier à l'aide des questions précédentes si c'est un palindrome ou non.