Programmation Procédurale : TD Chaînes de caractères

Consignes générales : N'oubliez pas pour ce TD comme pour les suivants de vous créer un répertoire consacré au TD et d'enregister vos codes desssus.

On rappelle que les commandes à taper dans le terminal pour compiler puis éxécuter votre programme C :
— Pour compiler : *gcc -o nom_executable nom_programme.c*
— Pour exécuter : *./nom_executable*
Il est conseillé de toujours écrire l'algorithme en pseudo-code avant de passer sur machine.

Exercice 1 (*Majuscule / minuscule*)

1. Écrire une fonction `char toUpper (char c)` qui prend en paramètre une lettre minuscule et retourne son équivalent en majuscule.
2. Écrire une fonction `char toLower (char c)` qui prend en paramètre une lettre majuscule et retourne son équivalent en minuscule.
3. Écrire une fonction `char changeLetter (char c)` qui
   — retourne l'équivalent de c en minuscule si c est une majuscule
   — retourne l'équivalent de c en majuscule si c est une minuscule
   — ne fait rien si le caractère n'est pas une lettre.
4. Dans le programme principal déclarer une chaîne de caractères suffisament grande pour stocker un prénom.
5. Saisir et afficher le prénom saisi.
6. D'après les règles de grammaire, la première lettre d'un prénom doit être une majuscule et les autres lettres des minuscules. Modifier la chaîne de caractère afin que le prénom saisi respecte ces règles.

Exercice 2 (*Fonctions de bases*)
Ne pas oublier de tester ces fonctions !

1. Écrire une fonction `getStringSize` qui prend en paramètre une chaîne de caractères et retourne sa taille.
2. Écrire une fonction `areStringEqual` qui prend en paramètre deux chaîne de caractères et qui retourne 1 si les chaînes sont les mêmes, 0 sinon.
3. Ecrire une fonction `concatStrings` qui prend en paramètre deux chaînes de caractères qui vont être concaténées (on va "coller" les chaînes de caractères l'une à la suite de l'autre).

Exercice 3 (*Code secret et cryptographie*)
Dans cet exercice, nous allons travailler sur des chaînes de caractères pour les modifier afin de chiffrer des messages.
Pour parvenir à déchiffrer des messages codés il faut deux informations : le message chiffré et une *clef privée*, permettant de définir la façon dont le message va être déchiffré. Nous allons donc écrire ces fonctions de chiffrement et de déchiffrement de différents messages.

On rappelle que les chaînes de caractères en C sont des tableaux de caractères.
Pour cet exercice, on pourra utiliser la fonction `strlen()` qui donne la taille d'une chaîne de caractères passée en paramètre. Cette fonction se trouve dans la bibliothèque `string.h`. **Pour plus de facilité, les messages à coder ne devront comporter que des minuscules et les espaces auront été supprimés.**

1. Le code de **César** repose sur un principe simple : on identifie chaque lettre du message original dans l'alphabet, on se décale d'un certain entier et on remplace la lettre originale par la nouvelle lettre obtenue. Si on parvient au bout de l'alphabet, on recommence au début).

   Par exemple : avec un décalage de 3, `a` devient `d`, `b` devient `e`, etc...
   La chaîne : `" hakunamatata"` avec un décalage de 3 devient `" kdnxqdpdwdwd"`
   C'est un chiffrement facile à réaliser mais aussi très facile à décoder.
   (a) Écrire une procédure `cesar_cipher(char message[], int shift)` qui modifie le tableau de caractères passé en paramètres en décalant chacune de ses lettres du paramètre `shift`.
   (b) Tester la procédure en chiffrant un message saisi dans le programme principal.
2. Écrire une procédure `cesar_decipher(char secret[], int shift)` qui prend en paramètre un tableau de caractères à décoder et le décalage avec lequel le message a été encodé (c'est la clef privée). Cette procédure modifie le contenu du tableau de caractères pour retrouver le message original.

3. Le code de chiffrement précédent est facile à trouver car chaque lettre du message original a toujours le même équivalent dans le message encodé. Vigenère a introduit une autre méthode de chiffrement. Cette méthode effectue des décalages qui ne sont pas toujours les mêmes et sont basés sur une clef privée qui est un mot complet.

   Le principe est le suivant : pour chaque lettre du message, on effectue le décalage qui correspond à chaque lettre de la clef privée.

   On pourra consulter (*https* : *//fr.m.wikipedia.org/wiki/Chiffre_de_Vigenere*)

   Voici un exemple :

   `Message à coder: "tuesunsorcierharry"`
   `Clef privée: "magie"`
   Voici comment va fonctionner l' encodage avec cet exemple :

t	u	e	s	u	n	s	o	r	c	i	e	r	h	a	r	r	y
m	a	g	i	e	m	a	g	i	e	m	a	g	i	e	m	a	g
f	u	k	a	y	z	s	u	z	g	u	e	x	p	e	d	r	e

   - *La première lettre du message est "t".*
   - *La première lettre du mot clef est "m".*

   La lettre "m" est la treizième lettre de la l'alphabet donc la lettre numéro 12, donc on décale de 12 : on arrive à "z" en avançant de 6, on revient au début et on atteint la lettre "f" en avançant de 6 lettres.

   On remplace donc la lettre "t" par la lettre "f".

   Pour la lettre suivante, comme la lettre du mot clef est "a", on décale de 0.

   Et ainsi de suite ...

   (a) Ecrire une procédure `vigenere_cipher( char message [], char keyword[] )` qui prend le message à encoder en paramètre et la clef d'encodage. Cette procédure modifie le tableau message pour modifier chaque caractère selon l'encodage de Vigenère.

   La tester avec l'exemple donné pour vérifier qu'elle fonctionne bien.

   (b) Écrire une procedure pour décoder un message lorsqu'on dispose de la clef.

4. Modifier le codage de Vigenère pour que l'on puisse utiliser des mots que l'on séparera avec un "_" ainsi que des majuscules. ex : "Tu_es_un_sorcier_Harry".
5. Écrire une procédure permettant de décrypter un message chiffré avec le code de César. Vous pouvez procéder par force brute (tester toutes les combinaison possible une par une) ou bien plus intelligement. Pour celà, On s'intéressera notamment à la fréquence d'apparition des lettres du message codé. Par exemple, il y a de bonnes chances que la lettre qui apparaisse le plus dans le message soit originellement la lettre "e".
   L'algorithme ne devra pas valider lui même sa proposition mais proposera diverses traductions à l'utilisateur.