Langage C
TP n° 6 : Chaînes de caractères

Les exercices indiqués avec une ∗ sont à rendre.

Important : Une chaîne de caractères est une suite de caractères se terminant avec le caractère nul ('\0'). Cette suite de caractères peut être stockée dans un tableau ou plus généralement dans une zone mémoire allouée par un `malloc` et dont l'adresse est stockée dans un pointeur sur `char` (donc un `char *`). **Il y a en C plusieurs fonctions de manipulation de chaînes, mais pour ce TP, nous n'autorisons, sauf mention explicite, que la fonction `strlen`**. L'allocation de zones mémoire se fera avec `malloc` et les recopies se feront avec `memmove`.

Exercice 1 : Opérations de base

1. Écrire une fonction `char * dupliquer(const char * s)` qui duplique une chaîne de caractères pointée par `s` et renvoie l'adresse de la nouvelle zone mémoire où une copie de `s` a été stockée.

2. Écrire une fonction `int ordrealpha(const char * s1, const char * s2)` qui prend deux pointeurs sur des chaînes ne contenant que des lettres en argument et retourne 1 si la chaîne d'adresse `s1` est supérieure dans l'ordre alphabétique à celle de `s2`, -1 si c'est l'inverse et 0 si les deux chaînes sont identiques.

3. Écrire une fonction `char * multiplier(const char *s, unsigned int n)` qui retourne un pointeur sur une nouvelle chaîne correspondant à la concaténation de `n` fois la chaîne pointée par `s`.

Exercice 2 : Passer des arguments à la fonction `main`

Il est possible de passer des arguments de la ligne de commande à un programme via la fonction `main`. On peut pour cela donner à `main` la signature `int main(int argc, char * argv[])`. `argc` est le nombre d'arguments et `argv` un tableau (de `argc` élments) qui contient les arguments. Le premier élément du tableau (`argv[0]`) est le nom de l'exécutable, et les éléments suivant sont les arguments passés à la ligne de commande. Il faut donc s'assurer que l'utilisateur passe le bon nombre d'arguments.

1. Modifier le code de l'exercice précédent pour invoquer la fonction `ordrealpha` sur deux chaînes passées en argument de la fonction `main`.
Exemple :

```
./executable hello world
hello < world
./executable foo foo
foo = foo
./executable hello
Ce programme attend 2 arguments, mais vous ne lui en avez fourni que 1.
```

2. Modifier le code de l'exercice précédent pour invoquer la fonction `multiplier` sur une chaîne et un entier passés en argument de la fonction `main`. Pour convertir une chaîne en un entier (lorsque c'est possible !), on pourra utiliser la fonction `int atoi(const char *str)`. Exemple :

```
./executable ab 5
ab x 5 ? abababababab
./executable ab ab
Ce programme attend un entier comme second argument.
./executable ab
Ce programme attend 2 arguments, mais vous ne lui en avez fourni que 1.
```

Exercice 3 : Algorithmique du texte (∗)

Dans cet exercice, on travaille sur une représentation de l'ADN sous forme de chaînes de caractères composées des caractères a, c, g, t. On nomme *mutation* une différence d'une chaîne par rapport à une autre de même taille. Par exemple, dans `"acca"`, `"cc"` à l'indice 1 est une mutation de longueur 2 par rapport à la chaîne `"aaaa"`. Une mutation est représentée par la structure suivante :

```c
typedef struct {
  size_t indice;
  size_t len;
} mutation;
```

1. Écrire une fonction `int nboc(const char *s, const char *sub)` qui renvoie le nombre d'occurrences de la chaîne d'adresse `sub` dans celle d'adresse `s`. Par exemple `"aa"` a trois occurrences dans `"aaacaa"` (aux positions `0`, `1` et `4`).
2. Écrire une fonction `mutation diff(const char *s, const char *t)` qui renvoie la première mutation de `t` par rapport à `s` (on vérifiera que les deux chaînes pointées sont de même longueur). Par exemple, si `m = diff("acca", "aaaa")`, alors `m.indice = 1` et `m.len = 2`. Si les chaînes sont identiques, la mutation renvoyée est de longueur nulle et d'indice quelconque.
3. En se servant de cette fonction, écrire `mutation longest(const char *s, const char *t)` qui renvoie la première plus longue mutation de `t` par rapport à `s`. Par exemple, si `m = longest("atcgatatt", "aaagccata")`, alors `m.indice = 1` et `m.len = 2`. Pour cela, on utilisera `diff` sur `s` et `t` puis à nouveau sur les suffixes de ces chaînes commençant après la première mutation pour trouver la deuxième, et ainsi de suite.
4. Écrire une fonction `char *longest_string(const char *s, const char *t)` qui renvoie l'adresse d'une chaîne qui est une copie de la plus longue mutation de `t` par rapport à `s`.