Feuille de TD - I

Exercice 1 (Comparaison entre chaînes de caractères)**.**
Ci-dessous, on suppose que les chaînes ne contiennent que des lettres.

(1) Écrire une fonction `int strcmp(string s1, string s2)` prenant en paramètres deux chaînes de caractères et qui renvoie 0 si le deux chaînes sont égales, -1 si la première chaîne précède la seconde dans l'ordre du dictionnaire ("ordre lexicographique"), et 1 sinon. On suppose que toutes les lettres sont en minuscule.

(2) Même question mais on suppose maintenant que les deux chaines comportent aussi bien des majuscules que des minuscules. Votre fonction de comparaison ne doit pas distinguer la casse des caractères. Les deux chaînes ne doivent pas être modifiées.

Exercice 2 (Conversion entre chaînes et entiers)**.**
On suppose les chiffres rangés consécutivement dans l'alphabet, de même que les lettres. La position relative des lettres par rapport aux chiffres est inconnue.

En C++, un nombre dont l'écriture commence par `0x` représente un nombre écrit en base 16, à l'aide des chiffres de `'0'` à `'9'` et des lettres de `'A'` à `'F'` (majuscules uniquement). S'il commence par `'0'` mais n'est pas suivi de `'x'`, on considère qu'il est en base 8, formé des chiffres de `'0'` à `'7'` uniquement. Sinon (y compris un `'0'` isolé), on considère qu'il est écrit en base 10.

(1) Écrire une fonction `int base(string s)` prenant en paramètre une chaîne de caractères représentant un entier **positif ou nul** et renvoie la base (parmi les trois ci-dessus) dans laquelle est écrit ce nombre, ou -1 si l'écriture du nombre est incorrecte.
Exemples : `"0"` et `"123"` représentent des nombres en base 10, `"0123"` un nombre en base 8, `"0x123"` un nombre en base 16. L'écriture`"0x2AC"` est correcte et désigne un nombre en base 16 ; l'écriture `"0129"` est incorrecte car `'9'` est illégal en base 8.

(2) Expliquez la différence entre les en-têtes de fonctions `f(string s)`, `f(string &s)` et `f(const string &s)`. Quand utilise-t-on un passage de paramètre "par référence" plutôt qu'un passage de paramètres "par valeur" ?

(3) Écrire une fonction `void int2string(int nombre, string &t)` qui prend en entrée un entier positif ou nul et une référence sur une chaîne de caractères et stocke dans la chaîne la représentation en base 10 de l'entier, les chiffres de poids fort à gauche. Par exemple si on passe en entrée le nombre `1280`, en sortie la chaîne doit contenir `"1280"`.

On fournira deux versions : l'une stocke le résultat dans l'ordre inverse avant de le retourner, l'autre stocke les caractères directement dans le bon ordre.

Exercice 3 (fusion de tableaux triés)**.**
Écrire une fonction qui prend en entrée deux vecteurs d'entiers `T1`, `T2` d'entiers **triés en ordre croissant** et qui renvoie un vecteur d'entiers, lui aussi trié aussi en ordre croissant, contenant les éléments de `T1` et `T2`. On profitera bien sûr du fait que `T1` et `T2` sont déjà triés. En-tête : `vector<int> fusion(vector<int> T1, vector<int> T2)`. Donnez-en deux versions qui diffèrent dans la manière de contrôler la boucle principale du programme : dans le premier cas la boucle principale est parcourue tant que les deux vecteurs n'ont pas été entièrement parcourus ; dans la seconde version on quitte la boucle principale dès que l'un des deux vecteurs a été entièrement parcouru.

Exercice 4 (recherche de facteurs)**.**
Écrire une fonction `size_t sousTableau(vector<int> t1, vector<int> t2)` qui prend en entrée deux vecteurs d'entiers `t1` et `t2` et teste si `t2` est un facteur de `t1` (i.e. tous les éléments de `t2` apparaissent **consécutivement** dans `t1` à partir d'un indice). La fonction renvoie le premier indice dans `t1` où cette situation se produit, et la longueur de `t1` dans le cas contraire. Pourquoi est-ce une valeur adaptée ? La constante `-1` le serait-elle aussi ?

Modifiez l'en-tête de la fonction de façon qu'elle soit réutilisable pour trouver à tour de rôle toutes les occurrences dans `t1` où `t2` est facteur : la recherche de la prochaine occurrence ne commence plus forcément au début de `t1`. Écrire une fonction qui imprime tous les indices de `t1` où débute une occurrence de `t2` (ces occurrences peuvent se chevaucher !). Par exemple, le vecteur `{ 0, 1, 2, 3, 4, 5, 6, 7, 5, 6, 7, 5, 6, 5, 5, 6 }` comporte deux occurrences du vecteur `{ 5, 6, 7 }` aux indices 5 et 8.

Exercice 5 (Compression naïve de chaînes)**.**
On veut compresser une chaîne de caractères en remplaçant toute séquence de `n` occurrences successives d'un même caractère, pour `n` compris entre 3 et 9, par ce caractère suivi de la représentation de `n`. S'il y a plus de 9 occurrences successives, on découpe en tranches d'au plus 9 occurrences. Ainsi la compression de `"aaaabbcddddddddddddddeee"` est `"a4bbcd9d4e3"`. On fera attention à écrire la représentation de `n` en tant que caractère et non pas comme entier.

(1) Écrire une fonction `size_t longueur(string s)` qui prend en entrée une chaîne compressée `s` et renvoie le nombre de caractères qu'aurait sa version décompressée.

(2) Écrire une fonction `string decompresse(string t1)` qui prend en entrée une chaîne compressée `t1` et renvoie sa version décompressée.

(3) Écrire une fonction `string compresse(string t1)` qui prend en entrée une chaîne `t1` formée de lettres minuscules et renvoie sa version compressée.

(4) (*facultatif*) Écrire une fonction `bool égale(string s, string s2)` qui prend en entrée deux chaînes compressées `s1` et `s2` et, **sans les décompresser,** renvoie `true` ssi `s1` et `s2` se décompresseraient en la même chaîne. On ne suppose pas que les deux chaînes ont été compressées de la même façon, juste qu'elles respectent le principe de compression. Ainsi, `"a4bbcd9d4e3"` et `"aaaabbcd7d6eee"` représentent la même chaîne décompressée.

Exercice 6 (Chiffrement)**.**
Dans les questions ci-dessous on veut crypter un texte stocké dans une string. Le texte est constitué uniquement de lettres majuscules et du caractère `' '` (espace). La fonction à écrire prend en paramètres le texte d'entrée, une seconde string qui contiendra en sortie le texte crypté ainsi qu'un troisième argument qui dépend de la méthode de cryptage.

(1) *(« méthode de César »)* Les lettres sont cryptées par un décalage décrit par un entier `n` (supposé positif ou nul) : à `'A'` on associe la lettre à la position `'A'+n` dans l'alphabet, à `'B'` la lettre à la position `'B' + n`, etc. Les décalages sont circulaires : si on dépasse `'Z'` on continue à partir de `'A'`. *Exemple* : avec un décalage de 3, *`CRYPTEZ CE TEXTE`* devient : *`FUBSWHC FH WHAWH`*. Écrire une fonction qui réalise ce chiffrement, d'en-tête :

```
void cesar(const string &entree, string &sortie, int n)
```

(2) *(« méthode par substitution »)* Le cryptage se fait via un tableau de 26 cases qui donne la substitution à appliquer à chaque lettre. **La fonction doit vérifier que la substitution décrit bien une bijection**, c'est-à-dire comporte chaque lettre une fois et une seule. Si ce n'est pas le cas, le texte de sortie est indifférent et la fonction renvoie *false*, sinon elle applique la substitution au texte d'entrée et renvoie *true*.

Exemple : avec la substitution *POIUYTREZAMLKJHGFDSQNBVCXW*, la chaîne *ASSEZ* devient *PSSYW*, la substitution associant `A` à `P`, `B` à `O`, `S` à `S`, `E` à `Y` et `Z` à `W`. En-tête :

```
bool subst(const string &entree, string &sortie, const string &subst) ;
```

(3) *(« Méthode de Vigenere »)* Il s'agit d'une méthode à décalage variable : le décalage d'une lettre varie selon sa position dans le texte et ses diverses occurrences ne seront pas toujours cryptées de la même manière. Les décalages à appliquer sont indiqués par une clef secrète (le troisième paramètre de la fonction) qui ne comporte que des lettres. Chaque lettre de la clef indique le décalage (exprimé par rapport à la lettre `A`) à appliquer à la lettre courante du texte. Si la lettre de la clef vaut `A`, le décalage à appliquer vaut `0`, si elle vaut `B` le décalage vaut `1`, etc. Le texte à crypter est découpé en blocs de n lettres (les espaces resteront inchangés), où n est la taille de la clef, puis chaque bloc subit la transformation suivante : à la première lettre du bloc on applique le décalage indiqué par la première lettre de la clef, à la deuxième lettre du bloc on applique le décalage indiqué par la deuxième lettre de la clef, etc. Quand on a épuisé la clef, on recommence à partir de son début pour le bloc suivant.

Exemple : avec le texte `"CSSEZ DE CRYPTER"` et la clef `"MACLE"` on obtient en résultat `"OSUPD PE ECCBTGC "` ; la première lettre de la clef indique un décalage qui substitue `M` à `A`, donc le premier `C` du texte est codé avec ce décalage et donne `O`, la seconde lettre de la clef associe `A` à `A`, donc la deuxième lettre du texte (le premier `S`) est inchangée, la troisième lettre associe `C` à `A`, donc le second `S` du texte est codé par `U`, etc. En-tête :

```
int vigenere(const string &entree, string &sortie, const string &clef)
```