INF1005C – PROGRAMMATION PROCÉDURALE
Travail dirigé no 6
Représentation des données

Objectif : Maitriser la représentation interne des données.

Durée : Deux séances de laboratoire.

Remise du travail : Avant 23h30 le mardi 18 juin.

Travail préparatoire : Lecture des exercices et de la documentation fournie et rédaction des algorithmes.

Directives :

- N'oubliez pas les entêtes de fichiers ni, aux endroits appropriés, les commentaires dans le code.
- Vous devez ajouter un en-tête pour chaque fonction. Dans l'écriture de l'entête d'une fonction, ne pas oublier de donner la description IN/OUT pour chacun des paramètres.
- Vous devez éliminer ou expliquer tout avertissement donné par le compilateur (au niveau d'avertissement /W4).
- Respectez le guide de codage (même points qu'au TD5) et le principe DRY (« don't repeat yourself »); vous pouvez ajouter autant de fonctions et structures que vous voulez pour que le programme soit facile à lire et sans duplication de code.
- **Il est interdit d'utiliser les variables globales**, sauf celles en lecture seule (constantes).

Documents à remettre : sur le site Moodle des travaux pratiques, vous remettrez l'ensemble des fichiers .cpp et .hpp compressés dans un fichier .zip en suivant la procédure de remise des TDs.

Mise en contexte

Le standard Unicode permet d'échanger des textes écrits dans toutes les langues partout dans le monde. L'Unicode est le standard d'échange textuel de loin le plus utilisé, et l'intégrer dans un programme permet de pouvoir traiter des caractères de toutes les langues en même temps dans le même programme. En Unicode, chaque caractère possède un numéro unique l'identifiant, appelé un point de code, et il existe plusieurs formats pour encoder ces valeurs dans un texte.

Les deux principaux formats d'encodage sont l'UTF-16 et l'UTF-8 (Universal Transformation Format). Comme les noms le disent, l'UTF-16 et l'UTF-8 utilise des unités, ou codets, de 16 et 8 bits, respectivement, pour encoder l'information.

Dans ce laboratoire, vous allez devoir écrire les fonctions qui permettent de passer de l'UTF-8 à l'UTF-16 et vice-versa.

Matériel fourni

En plus du code, on vous donne aussi trois textes encodés en UTF-8 pour faire vos tests :

1. *TroisMousquetaires.txt* : Texte en français, donc caractères latins
2. *Katyusha.txt* : Chanson russe, donc caractères cyrilliques
3. *CaracteresAnciens.txt* : Anciens caractères occidentaux, prennent deux codets en UTF-16

Présentation du travail à réaliser

Tout d'abord, lisez bien les articles Wikipédia qu'on cite. Les exemples qui y sont faits peuvent vous être très utiles à la compréhension des tâches à accomplir.

Vous avez plusieurs fonctions à implémenter dans *utf8.cpp*, toutes assez longues. On vous fournit les squelettes dans *utf8.cpp* et les prototypes dans *utf8.hpp*, ainsi que des constantes dans *constantes.hpp*. On vous fournit aussi les fichiers *utf16.hpp* et *utf16.cpp* qui font la même chose que ce qu'on vous demande, mais pour l'UTF-16. Vous avez donc avantage à bien lire le code de ces derniers, car ils peuvent être très utiles pour votre compréhension des tâches à faire. Pour chaque fonction, vous devez remplir les *TODO* qui vous sont donnés dans le code. Vous pouvez écrire plus de fonctions si vous le voulez, mais ce n'est pas nécessaire. Vous devez ensuite écrire le `main` de la même façon en suivant les *TODO*.

On vous fournit aussi les fichiers *debogageMemoire.hpp* et *unicode.hpp*, qui servent respectivement à faire la détection de fuites de mémoire, et à initialiser la console pour l'écriture en Unicode dans `wcout`. Voir l'annexe 1 pour plus d'infos.

N'oubliez pas d'écrire votre documentation!

Présentation du format UTF-8

en.wikipedia.org/wiki/UTF-8 : Lire la section *Description* avec les exemples

L'UTF-8 utilise des codets de 8 bits et fut conçu pour maintenir la rétrocompatibilité avec l'ASCII. Un point de code (un caractère) est encodé sur un à quatre octets. Les caractères ASCII peuvent aller tels quels sur un octet sans encodage différent, les autres caractères sont encodés sur plusieurs octets selon une structure particulière. Dans le tableau suivant, les *x* représentent les bits de point de code.

Nb d'octets	Nb de bits pour le code	Octet 1	Octet 2	Octet 3	Octet 4
1	7	`0xxxxxxx`			
2	11	`110xxxxx`	`10xxxxxx`		
3	16	`1110xxxx`	`10xxxxxx`	`10xxxxxx`	
4	21	`11110xxx`	`10xxxxxx`	`10xxxxxx`	`10xxxxxx`

Un texte en UTF-8 peut être écrit tel-quel dans un fichier, sans avoir besoin d'entêtes ou de marques, mais certains programmes demandent que le point de code spécial BOM (voir les constantes dans CodeFourni.hpp) soit encodé en UTF-8 au début du fichier pour identifier l'encodage utilisé.

Présentation du format UTF-16

en.wikipedia.org/wiki/UTF-16 : Bien lire la section *Description* avec les exemples

L'UTF-16 encode les caractères sur des codets de 16 bits, donc deux octets. Les points de code qui entrent dans 16 bits (< 0xFFFF) sont écrits directement sur deux octets, sans préfixes. Les points de code dépassant cette plage sont encodés sur deux codets de 16 bits, donc 4 octets.

Vous remarquerez qu'on utilise des `u16string` dans le code pour travailler avec l'UTF-16. Les `u16string` sont comme des `string`, mais avec des `char16_t` au lieu de `char`, donc des caractères de 16 bits.

Pour encoder les points de code à plusieurs codets, il faut suivre la procédure suivante :

1. On soustrait 0x10000 au point de code, nous laissant un nombre de 20 bits
2. Les dix bits de poids fort de ce nombre à 20 bits sont extraits et additionnés à 0xD800 (constante `MULTI_CODET_UTF16_HAUT` dans le code)
3. Les dix bits de poids faible du nombre sont extraits et additionnés à 0xDC00 (constante `MULTI_CODET_UTF16_BAS` dans le code)
4. Le codet avec les bits hauts (étape 2) vient en premier dans la string où il se trouve (indice plus faible), l'autre vient après.

Lorsque vient le temps d'écrire une string en UTF-16 dans un fichier, il faut d'abord écrire une marque d'ordonnancement des octets (*byte order mark*) au début du fichier. Cela correspond à la constante `BOM` dans le code fourni. On écrit donc le BOM au début, puis les codets de 16 bits dans l'ordre indiqué à l'étape 4.

Dans le TD, nous allons seulement traiter de l'UTF-16 petit-boutiste (*little-endian* en anglais) et ne traiteront pas le *big-endian*.

Annexe 1 : Utilisation des outils de programmation et débogage.

Utilisation d'Unicode :
Pour afficher des caractères accentués, vous devez utiliser les versions « wide » de cout, string, et char, qui sont respectivement `wcout`, `wstring`, et `wchar_t`. Les chaînes et caractères « wide » s'écrivent avec un L majuscule devant, tel que L"Allô" et L'ô'. Des chaînes et caractères ordinaires peuvent aussi être affichés sur `wcout` mais ne doivent pas contenir de caractères accentués. Un exemple d'affichage se trouve dans le programme fourni.

Rapport sur les fuites de mémoire et la corruption autour des blocs alloués :
Le programme inclus des versions de débogage de « `new` » et « `delete` », qui permettent de détecter si un bloc n'a jamais été désalloué, et afficher à la fin de l'exécution la ligne du programme qui a fait l'allocation. L'allocation de mémoire est aussi configurée pour vérifier la corruption à chaque allocation, permettant d'intercepter des écritures hors bornes d'un tableau alloué. Si une corruption est détectée, le débogueur affichera un message indiquant une possible « défaillance du tas », et la fenêtre « Pile des appels » vous permettra de voir à quel endroit la détection a été faite (regardez vers le bas de la pile d'appels pour l'élément le plus haut qui correspond à votre programme, et non ceux au dessus qui correspondent aux fonctions de C++). Si vous avez un problème de corruption, vous pouvez utiliser « `_CrtCheckMemory()` » à n'importe quelle ligne pour vérifier si la corruption arrive avant ou après cette ligne (la fonction retourne faux s'il y a corruption). Il est aussi possible d'exécuter la vérification dans le débogueur; dans la liste des espions (fenêtre « Espion 1 »), ajoutez l'espion « `((int(*)())ucrtbased.dll!_CrtCheckMemory)()` ». La valeur sera 0 s'il y a corruption, 1 sinon, et il suffit de cliquer sur les flèches en rond pour revérifier la corruption à n'importe quel moment pendant qu'on trace dans le débogueur.

Utilisation de la liste des choses à faire :
Le code contient des commentaires « TODO » que Visual Studio reconnaît. Vous devez premièrement activer
l'option, dans le menu Outils > Options…, allez dans Éditeur de texte > C/C++ > Mise en forme > Divers > Énumérer les tâches de commentaire, pour mettre cette option à « True », puis OK. Pour afficher la liste, allez
dans le menu Affichage, sous-menu Autres fenêtres, cliquez sur Liste des tâches. Choisissez ensuite dans cette fenêtre d'utiliser les commentaires. Vous pouvez double-cliquer sur les « TODO » pour aller à l'endroit où il se trouve dans le code. Vous pouvez ajouter vos propres TODO en commentaire pendant que vous programmez, et les enlever lorsque la fonctionnalité est terminée.

Annexe 2 : Points du guide de codage à respecter

Les points du **guide de codage** à respecter **impérativement** pour ce TD sont les mêmes que pour le TD4 :
(voir le guide de codage sur le site Moodle du cours pour la description détaillée de chacun de ces points)

Points du TD5 :

2 : noms des types en `UpperCamelCase`
3 : noms des variables en `lowerCamelCase`
5 : noms des fonctions en `lowerCamelCase`
21 : pluriel pour les tableaux (`int nombres[];`)
22 : préfixe *n* pour désigner un nombre d'objets (`int nElements;`)
24 : variables d'itération `i`, `j`, `k` mais jamais `l`
27 : éviter les abréviations (les acronymes communs doivent être gardés en acronymes)
29 : éviter la négation dans les noms
33 : entête de fichier
42 : `#include` au début
46 : initialiser à la déclaration
47 : pas plus d'une signification par variable
48 : aucune variable globale (les constantes globales sont tout à fait permises)
50 : mettre le & près du type
51 : test de 0 explicite (`if (nombre != 0)`)
52, 14 : variables vivantes le moins longtemps possible
53-54 : boucles `for` et `while`
58-61 : instructions conditionnelles
62 : pas de nombres magiques dans le code
67-78, 88 : indentation du code et commentaires
83-84 : aligner les variables lors des déclarations ainsi que les énoncés
85 : mieux écrire du code incompréhensible plutôt qu'y ajouter des commentaires
89 : entêtes de fonctions; y indiquer clairement les paramètres [out] et [in,out]