IFT1166 – TRAVAIL PRATIQUE #3 – 05 juin 2007

Correcteur orthographique

Mohamed Lokbani

Équipes: le travail peut-être fait en binôme. Vous ne remettez alors qu'un travail par équipe.
Remise : une seule remise est à effectuer par voie électronique le jeudi **28 juin 2007, 23h59 au plus tard, sans possibilités de prorogation**.
Conseils: n'attendez pas le dernier jour avant la remise pour commencer. Vous n'aurez pas le temps nécessaire pour le faire.

But : ce TP a pour but de vous faire pratiquer l'utilisation des classes et des objets.

Description : dans ce TP, vous allez réaliser un correcteur orthographique simplifié. Il vous est demandé de compléter trois classes: « **Phrase** », « **Dictionnaire** » et « **Correcteur** » dont les spécificités sont décrites dans le présent document.

Recommandations : vous allez devoir compléter les parties de code qui sont proposées dans ce sujet. Les commentaires qui sont reportés dans ce code font partie intégrante du sujet et devront à ce titre être lus pour une compréhension complète du TP. Un canevas de votre code (c'est à dire le code que vous devez compléter) « **process.h** » est disponible; il reprend les différents points développés dans ce document.
Écrivez une classe à la fois, testez-la en écrivant votre propre programme principal. Le sujet vous propose des utilisations possibles des classes que vous devez programmer.

Les classes de ce travail : ce TP vise à vous faire programmer les 3 classes « Phrase », « Dictionnaire » et « Correcteur ».

- **Phrase** : cette classe va servir à coder une phrase. Elle va servir pour représenter dans un format « pratique » la phrase à corriger permettant ainsi d'accéder facilement à chacun de ses mots.
Une phrase est représentée par une table de « string ». Chaque élément de la table référant à un mot particulier de la phrase. La dimension de la table sera déterminée au moment de la construction.
C'est le constructeur de cette classe qui s'occupera de découper la phrase (objet string), passée en argument, en mots. Ces mots seront rangés dans une table (un membre interne) qui sera allouée de manière adéquate.
On considérera dans ce TP qu'une phrase est une suite de mots séparés par un blanc (caractère espace). Ainsi la déclaration suivante créera un objet dont la représentation interne sera une table de 5 mots (un mot étant ici un objet de type string):

```
Phrase *p = new Phrase ("Voici un exemple de phrase");
cout << "Nombre de mots dans la phrase: " << p->getNbMots());
cout << "2ème mot: " << p->getMot(1));
```

Tout objet de type Phrase pourra accéder à un mot quelconque de la phrase par la méthode « getMot(int) » ainsi qu'a son nombre de mots par la méthode « getNbMots() ».

Un squelette de la classe « Phrase » est fourni dans le fichier « process.h »

- Dictionnaire : elle sert à stocker un dictionnaire. Un dictionnaire est ici codé sous la forme d'une table de mots, c'est à dire une table de références sur des objets « string ». Les mots rangés dans un dictionnaire sont lus depuis un fichier texte dont le nom est passé en argument. Un fichier dictionnaire est simplement un fichier texte dont chaque ligne est un mot. On supposera qu'un tel fichier est trié en ordre alphanumérique croissant. Deux fichiers dictionnaires sont mis à votre disposition afin de tester votre travail.

- « dico200.dic » contient 200 mots
- « dico1000.dic » contient 1000 mots

Le contenu du fichier « dicoxxx.dic » est sauvegardé d'abord dans une instance (objet) de la classe « Donnees ». Cette classe est définie dans le fichier « data.h ». Ce fichier vous est fourni. Par la suite, cet objet de la classe « Donnees » est passé comme argument du constructeur de « Dictionnaire ».
Voici un exemple de code que votre programme devrait être capable de créer :

```
Donnees* mondico = new Donnees;
Dictionnaire *d = new Dictionnaire (mondico);
cout << "Nb de mots dans le dictionnaire:" << d->getNbMots());
cout << "Est-ce que le mot banane est dans le dico ? " << g->appartient("banane"));
```

Un squelette de la classe « Dictionnaire » est fourni dans le fichier « process.h »

Correcteur : un correcteur orthographique est un objet de la classe « Correcteur » que vous allez devoir définir en respectant les contraintes suivantes :
- La classe « Correcteur » admet au moins un constructeur prenant en argument un objet de la classe « Données ». Ceci permet par exemple de travailler sur des dictionnaires de différentes langues et donc de corriger des phrases de langues différentes.
- La classe « Correcteur » possède un objet membre du type « Phrase » qui contient la phrase à corriger.
- Un utilisateur de la classe « Correcteur » (probablement vous !) peut lancer une correction en invoquant la méthode « corrige(string p) » qui prend une phrase (objet string) en argument.

Voici comment un utilisateur pourrait utiliser cette classe:

```
string ph1 = "Voici une phrase de six mots";
string ph2 = "Voici une phrase plus courte";
Correcteur *c = new Correcteur (mondico);
c->corrige(ph1);
cout << ("La phrase " +ph1 + " contient " + c->getNbFautes());
cout << ("La phrase " +ph1 + " contient " + c->getNbMotsInconnusDuDictionnaire()));
c->corrige(ph2);
cout << ("La phrase " +ph2 + " contient " + c->getNbFautes());
cout << ("La phrase " +ph2 + " contient " + c->getNbMotsInconnusDuDictionnaire());
```

Un squelette de la classe « Correcteur » est fourni dans le fichier « process.h »

Les corrections traitées par votre correcteur : il vous appartient de coder au moins la détection d'erreurs concernant le dédoublement de consonnes. Voici quelques exemples que votre programme devrait être capable de détecter:

mot à corriger	correction	diagnostique
asiégez	assiégez	manque un s
courier	courrier	manque un r
cachotier	cachottier	manque un t
sacharifier	saccharifier	manque un c
vaccancier	vacancier	un c en trop
tariffier	tarifier	un f en trop

Pour ce faire, vous pouvez implanter l'algorithme suivant où M désigne le mot à corriger:

-1- Rechercher le mot « M » dans le dictionnaire. S'il existe alors le mot ne présente pas de faute. Fin

-2- Si le mot « M » contient une consonne double (ex: tariffier), créez un nouveau mot « M' » en éliminant la consonne double (ex: tarifier). Sinon allez en 4.

On supposera qu'un mot contient au plus une consonne double. Si cela n'est pas le cas, seul le premier doublement est traité. Votre correcteur ne sera donc pas capable de corriger le mot « vaccanccier » qui contient deux fautes de doublement.

-3- Si le mot « M' » est dans le dictionnaire, alors le mot « M » contient une erreur (une consonne double au lieu d'une consonne simple) et le mot « M' » est une correction possible. Fin

-4- Soit « nc » le nombre de consonnes non doublées dans le mot « M » (ex: 3 dans asiégez). Tentez tour à tour les « nc » transformations de « M » en « M' » en doublant à chaque fois la consonne considérée (ex: « M' » prendra tour à tour les valeurs: « assiégez », « asiéggez » puis « asiégezz »).

-5- Si « M' » est dans le dictionnaire, alors « M » contient une erreur (il manque une consonne) et « M' » est une correction possible. Fin

-6- Si aucune transformation n'a été fructueuse, alors le correcteur n'émettra pas d'hypothèse particulière et se contentera d'indiquer qu'il ne connaît pas le mot dont il avait la charge de la correction. Dans ce TP, ce cas sera très fréquent, par contre la couverture des dictionnaires mis à votre disposition est très faible.

Sorties du correcteur : Observons directement sur un exemple ce que doit afficher le programme suivant :

```
Correcteur* correcteur = new Correcteur (mondico);
correcteur->corrige("un vacancier cachottier parle à son hallebardier");
```

mot	correction	diagnostique
un	un	mot inconnu
vacanccier	vacancier	un c en trop
cachotier	cachottier	un t en trop
parle	parle	mot inconnu
à	à	mot inconnu
son	son	mot inconnu
halebardier	hallebardier	manque un l
La phrase contient 3 erreur(s)		

Vous devez respecter le formatage de l'affichage en sortie. Deux sorties vous sont fournies :

- « sortie.dico200 » pour le dico contenant 200 mots
- « sortie.dico1000 » pour le dico contenant 1000 mots

Vous pouvez utiliser la même technique de comparaison utilisée dans les précédents travaux (la commande diff) pour comparer les sorties que vous obtenez avec celles fournies en exemple.

Fichiers à remettre : « process.h » va contenir votre programme complété et un rapport expliquant la démarche suivie pour la réalisation de ce travail. Le rapport doit être au format « pdf » (voir là aussi la FAQ sur la page web du cours sur la façon de générer un fichier au format « pdf »).

Hypothèses et contraintes :

- pour commencer, vous devez déjà respecter les différentes contraintes de programmation précédemment décrites.
- IFT1166 est un cours en C++, donc votre programme doit être écrit en C++ et pas en C ni en Java! Nous n'autoriserons aucune référence au langage C. Par exemple, l'instruction [#include <stdio.h>] fait référence au langage C donc elle n'est pas autorisée. Utilisez plutôt [#include <iostream>].
- ce travail n'est pas un exercice d'algorithmique donc pas besoin de le compliquer pour rien!
- vous devez vous assurer de ne pas changer les noms des fichiers et respecter par la même occasion le format de l'affichage en sortie.

Remise : il est important de noter que votre TP sera compilé avec gcc3.2.4. Si par choix vous décidez d'utiliser un autre compilateur, vérifiez que le code que vous avez produit (devant normalement fonctionner correctement chez

vous) fonctionne aussi bien sur les ordinateurs de la DESI. Pour avoir la version du compilateur, utilisez la commande "**gcc –v**", qui devra donner le numéro de version "**3.2.4**".
Par ailleurs, assurez-vous de la présence de l'option « pedantic » sur la ligne de compilation. (Cette option n'est pas activée par défaut dans l'utilitaire « devcpp ». Il faudra donc penser à l'activer).
Si vous avez regroupé l'ensemble des fichiers dans un seul fichier « tp3.zip », vous devez faire la remise comme suit :

1. commencez d'abord par vous connecter sur la machine « remise » comme il a été pratiqué dans la démo #01.
2. envoyez l'ensemble de vos fichiers par la procédure de remise électronique habituelle (Pour obtenir de l'aide sur cette commande, tapez dans un Xterm : man remise). Respectez les noms des fichiers.

   remise ift1166 tp3 tp3.zip

3. Vérifiez que la remise s'est effectuée correctement.

   remise –v ift1166 tp3

Barème : ce TP3 est noté sur 15 points.

Compilation et respect des spécifications	**1**
Codage, style, efficacité etc.	**6**
Rapport	**3**
Tests (fournis et non fournis)	**5**

En plus du précédent barème, vous risquez de perdre des points dans les cas suivants ….

- La non remise électronique (volontaire ou par erreur) est sanctionnée par la note 0.
- Les programmes ne contenant pas d'en-tête, -1 point.
- Un programme qui ne compile pas : 0.
- Un programme qui compile mais ne réalise pas les choses prévues dans la spécification : 0.
- Les avertissements (warnings) non corrigés : cela dépend de la quantité! À partir de -0.25 et plus.
- Le non respect du nom du fichier entraîne une erreur de compilation donc un des points de la spécification n'a pas été respecté : 0.
- Aberration dans le codage : même si tous les chemins mènent à Rome, faites l'effort nécessaire pour éviter de prendre le plus long!

Automatisation de la correction : c'est « l'ordinateur » qui va corriger en quelque sorte vos travaux. Ce dernier va utiliser des scripts qui vont tenir compte au détail près des contraintes imposées. Pas d'espaces blancs en trop, pas de commentaires supplémentaires. Vous devez produire des sorties **identiques** à celles fournies.

Ainsi donc au moment de la remise de votre travail, vous avez déjà une idée approximative de la note que vous allez obtenir. Si la commande « diff » ne signale pas de différences entre les deux fichiers, vous êtes sur la bonne voie pour obtenir une note élevée.

Des questions à propos de ce TP?

Une seule adresse : pift1166@iro.umontreal.ca

Pour faciliter le traitement de votre requête, inclure dans le sujet de votre email, au moins la chaîne: [IFT1166] et une référence au **tp03**.

Mise à jour

05-06-2007 diffusion