IFT1169 – TRAVAIL PRATIQUE #1 – 25 janvier 2008

Le couteau suisse du C++!

Mohamed Lokbani

Équipes: le travail est à faire en monôme (**une seule personne**).
Remise : une seule remise est à effectuer par voie électronique le jeudi **7 février 2008, 23h59 au plus tard, sans possibilités de prorogation**.
Conseils: n'attendez pas le dernier jour avant la remise pour engager votre travail. Vous n'aurez pas le temps nécessaire pour le réaliser.

But : ce TP a pour but de vous familiariser avec votre environnement de travail d'une part et d'autre part de manipuler des gadgets de base nécessaires à un programmeur C++. Nous allons introduire pour cela l'utilisation de la classe string, permettant de manipuler avec une certaine aisance les chaînes de caractères; ainsi que la manipulation des arguments passés en ligne de commande et ce, afin de vous faire pratiquer la procédure de remise, la rédaction d'un rapport et le respect des échéances.

Énoncé : on vous demande d'écrire dans un même programme C++, les instructions nécessaires permettant de lire à l'entrée standard une série de mots passés en ligne de commande et de les afficher, par la suite, sur la sortie standard dans un format préalablement fixé.

Caractéristiques techniques :

- **Format d'affichage :** il correspond à l'une des justifications suivantes :
  - **g :** pour une justification à gauche.
  - **d :** pour une justification à droite.
  - **c :** pour une justification au centre.
- **Taille par défaut:** elle correspond au nombre total de caractères à afficher. Pour ce travail, la taille par défaut est fixée à 20 caractères.
- **Remplissage :** elle définit le caractère de remplissage. Pour ce travail, ce caractère est **#**.

Vous pouvez ajouter à votre aise d'autres éléments si vous le jugez nécessaire.

Scénarios entrée/sortie : Les éléments seront lus à partir de l'entrée standard. Nous allons d'abord préciser le format d'affichage (i.e. une des lettres « **g** », « **d** », « **c** ») puis la série de mots à lire. Ces paramètres sur la ligne de commande seront séparés par un espace blanc (voir le fichier fourni avec ce travail).

Après avoir lu ces informations, tout en tenant compte du format demandé, la série de mots sera affichée comme suit :

- si **g** : vous devez afficher la phrase à gauche et compléter à sa droite les champs vides par le caractère #.
- si **d** : vous devez afficher la phrase à droite et compléter à sa gauche les champs vides par le caractère #.
- si **c** : vous devez afficher la phrase au centre et compléter à sa droite et à sa gauche les champs vides par le caractère #. Si dans ce cas le nombre de champs vides est un nombre impair, le caractère # en trop sera placé à droite de la phrase.

Pour ce travail, la taille par défaut pour l'affichage en sortie a été fixée à 20 caractères, seulement. Or, il se peut que la série de mots ait une taille bien supérieure à la taille par défaut. Dans ce cas, il faut faire les réajustements nécessaires. La nouvelle taille (par défaut) doit être toujours un multiple de la taille par défaut. Par exemple, si la série de mots à une taille totale de 24 (ou 38, 44, 78) caractères, espaces blancs entre les mots compris, la taille par défaut sera donc portée à 40 (resp. 40, 60, 80) caractères.

Hypothèses et contraintes :

- pour commencer, vous devez déjà respecter les différentes contraintes de programmation précédemment décrites.
- « IFT1169 » est un cours en « C++ », donc votre programme doit être écrit en « C++ » et pas en « C » ni en « Java »! Nous n'autorisons 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 compliquer le travail 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.
- vous devez vérifier si les entrées sont correctes et afficher en conséquence les messages d'erreur appropriés (voir annexe « **1** ») :

*) le nombre d'arguments est correct, i.e. avoir obligatoirement en entrée le format approprié et la série de mots (devant contenir au moins un mot).

*) un des formats doit être obligatoirement g, d, ou c et autres exclus.

- il faudra vous assurer de respecter les noms des fichiers demandés, ainsi que le format de l'affichage en sortie.
- des fichiers d'entrée/sortie sont fournis. On ne vous demande nullement d'écrire du code pour lire/écrire à partir de fichiers. Vous pouvez tester vos programmes à la main (en insérant manuellement les informations) comme vous pouvez utiliser les redirections (plus faciles à manier et permettant d'éviter les erreurs de frappe). Les redirections sont au nombre de deux : **<** pour l'entrée et **>** pour la sortie. Exemple : [**tp1.exe < data.txt > masortie.txt**]. Dans cet exemple, la lecture se fera à partir du fichier « data.txt » et l'écriture se fera dans le fichier « masortie.txt ».
- les fichiers entrée/sortie ont été générés sous « MinGW/Msys », ils sont donc au format « DOS » étendu. Si vous travaillez sur « LINUX », il faudra vous assurer de les convertir au format « LINUX ». Pour cela, il faudra utiliser les deux commandes suivantes :

*) « **dos2unix** » pour convertir du format « DOS » au format « LINUX » (pour la conversion inverse, il faudra utiliser la commande « **unix2dos** »).

*) « **recode latin1..utf8 fichier** » pour passer du format « ISO-LATIN 1 » au format « UTF-8 ».

Ces commandes sont disponibles dans un « xterm » d'une plateforme « LINUX ».

- à signaler que nous utiliserons aussi d'autres fichiers pour la correction. Si vous avez respecté les contraintes imposées, peu importe le jeu de test utilisé, il devra fournir fatalement un résultat correct.

Remise : il est important de noter que votre TP sera compilé avec « gcc3.4.2 ». 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 « **3.4.2** ».

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 « tp1.zip », vous devez faire la remise comme suit :

1. commencez d'abord par vous connecter sur la machine « **casier** » 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 ift1169 tp1 tp1.zip

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

   remise –v ift1169 tp1

Barème : ce TP1 est noté sur 4 points.

Compilation et respect des spécifications	**0.75**
Codage, commentaires etc.	**1.25**
Rapport	**0.75**
Tests (fournis et non fournis)	**1.25**

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 une sortie **identique** à celle du fichier « **sortieref.txt** ».

Pour s'en assurer de cela, commencez d'abord par rediriger la sortie dans un fichier avec le symbole >. Comparez par la suite votre sortie avec la sortie de référence en utilisant pour cela la commande « diff » (sous MinGW/MSys) sinon la commande « fc » sous « DOS », ajustez juste les options. La commande « diff » permet donc de trouver les différences entre les fichiers. Si les fichiers sont identiques au caractère près, la commande ne retourne rien, sinon la liste des différences.

compilation	**g++ -Wall –pedantic[1] -o tp1.exe tp1.cpp**
redirection	**tp1.exe > masortie.txt**
comparaison	**`diff -b -w -i -B` sortieref.txt masortie.txt**

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 : dift1169@iro.umontreal.ca

Pour faciliter le traitement de votre requête, inclure dans le sujet de votre email, au moins la chaîne: [IFT1169] et une référence au **travail01**.

Mise à jour

25-01-2008 diffusion

[1] L'éditeur « Scite » prend en compte par défaut de ces options.

Annexe -1-

Pour chaque test, nous avons fourni la ligne de commande en entrée (colonne 1) et le résultat obtenu (colonne 2). Nous avons fourni aussi des explications complémentaires pour chacun des tests. À signaler que chaque sortie est affichée obligatoirement entre **' '.**

Test	Entrée	Sortie
1	tp1.exe g ceci est exemple	'ceci est un exemple#'
La lettre « g », donc une justification à gauche. De ce fait le « # » est à droite de la phrase.		
2	tp1.exe d ceci est exemple	'#ceci est un exemple'
La lettre « d », donc une justification à droite. De ce fait le « # » est à gauche de la phrase.		
3	tp1.exe c ceci est exemple	'ceci est un exemple#'
La lettre « c », donc une justification au centre. La phrase compte 19 caractères, espaces compris. La taille par défaut étant 20, un seul # est permis. Comme c'est un nombre impair, le # sera placé à droite de la phrase.		
4	tp1.exe c ceci est un autre exemple plus grand	'##ceci est un autre exemple plus grand##'
La lettre « c », donc une justification au centre. La phrase compte 36 caractères, espaces compris. La taille par défaut a été ajustée à 40 (2x20). 4 # sont permis pour compléter la chaîne. Comme c'est un nombre pair, 2 # à droite et à gauche de la phrase.		
5	tp1.exe ceci est un autre exemple plus grand	Erreur: justification inconnue!
Il y a erreur, car nous avons omis de préciser le format d'affichage.		
6	tp1.exe c	Erreur: le nombre d'arguments doit être au moins égal à 3!
Il y a erreur, car nous avons omis de préciser la série de mots.		

Annexe -2-

Si la manipulation des pointeurs vous perturbe pour ce premier travail, vous pouvez convertir les chaînes de caractères, i.e. « char* » vers le type « string ».

En C++ la classe « string » permet de manipuler les chaînes de caractères sans trop se soucier de la gestion de pointeurs qui est faite par derrière. Le programme suivant présente quelques exemples d'utilisation du type « string ». À noter que pour pouvoir utiliser « string », il faut impérativement inclure le fichier d'entête « <string> ».

```
#include <iostream>
#include <string>

using namespace std;

int main() {

    string chaine, deuxchaines; // définition de 2 variables
    char tab[5]={'a','b','c','d','\0'}; // une chaîne de caractères sous la forme d'un tableau

    chaine = "ceci"; // initialisation d'une chaîne
    cout << "chaine au départ: " << chaine << endl;

    string uneautre = " est un exemple"; // définition et initialisation les 2 à la fois
    cout << "une autre chaine: " << uneautre << endl;

    chaine = chaine + uneautre; // on ajoute deux chaînes, une addition classique
    cout << "chaine après ajout: " << chaine << endl;

    cout << "la taille de ma chaine: " << chaine.length() << endl; // pour avoir la taille

    string untableau(tab); // définition et initialisation à l'aide d'un tableau
    cout << "untableau: " << untableau << endl;

    return 0;

}
```

Sortie

```
>chaine
chaine au départ: ceci
une autre chaine:  est un exemple
chaine après ajout: ceci est un exemple
la taille de ma chaine: 19
untableau: abcd
>Exit code: 0
```