IFT1166 – TRAVAIL PRATIQUE #1 – 15 mai 2007

Pointeurs et calculette!

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 **07 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 est composé de deux exercices.
L'exercice -1- va vous permettre de pratiquer la notion de pointeurs.
L'exercice -2- consiste à la réalisation d'un programme complet en C++ en n'utilisant que les notions du non orienté objet de ce langage, comme par exemple les fonctions.

Exercice -1- : expliquez en **détail** la valeur obtenue par la variable « Unchar » pour chacune des instructions (1 à 10) du fichier « exo1tp2h07.cpp ».

Fichiers à remettre : votre réponse doit être consignée dans le fichier « pdf » « exo1tp2h07.pdf ». Voir la « FAQ » sur la page web du cours sur la façon de générer un fichier au format « pdf ».

Exercice -2- : écrivez un programme « C++ » permettant d'effectuer les opérations d'addition et de soustraction sur des chiffres romains. Votre programme prend en entrée une opération mathématique en chiffres romains et affiche en sortie le résultat de cette opération en chiffres romains. Par exemple:

MMI - MCMLXX + X = 2001 - 1970 + 10 = 41 = XLI

Description des chiffres romains : les valeurs décimales inférieures à 3999 sont représentées dans le système numérique romain par des lettres. Le tableau suivant donne la correspondance entre le système décimal et le système romain :

Chiffre (lettre) dans le système romain	correspondance dans le système décimal
M	1000
D	500
C	100
L	50
X	10
V	5
I	1

Calcul dans le système romain : les chiffres romains sont écrits dans un ordre descendant. Les chiffres sont additionnés pour produire la valeur finale, sauf pour certains préfixes (voir plus bas).

Nombre d'occurrences :
1) Pas plus de 3 occurrences de M, C, X ou I peuvent apparaître consécutivement dans un nombre romain, et non plus qu'un seul D, L, ou V dans ce nombre là. Ainsi par exemple, on peut avoir MMM mais pas MMMM et MCCC, mais pas MCCCC etc.
2) De même, le chiffre C, X ou I peut être utilisé comme préfixe, et sa valeur est donc retranchée au lieu d'être ajoutée au nombre, comme suit:

Un C qui préfixe un M ou un D, représente respectivement 900 ou 400, et les chiffres M & D sont écrits après le C ; par exemple:

CM <=> 900
Car M=1000 et C=100.
Avoir C avant M, on retranche => cela signifie 1000-100=900.

3) Les chiffres qui viennent après le M ou le D, ne peuvent totaliser plus de 99. On ne peut pas avoir donc: CMM.

4) Un X qui préfixe un C ou un L, représente respectivement 90 ou 40, et les chiffres C & L sont écrits après le X ; par exemple:

XC <=> 90
Car C=100 et X=10.
Avoir X avant C, on retranche => cela signifie 100-10=90.

5) Un I qui préfixe un X ou un V, représente respectivement 9 ou 4, et les chiffres X & V, sont écrits après le I à la fin du chiffre, par exemple:

IV <=> 4
Car V=5 et I=1.
Avoir I avant V, on retranche => cela signifie 5-1=4.

6) Ainsi la valeur (en romain) de MMI correspond à 2001 en décimal, MCMLXX à 1970 etc.

Comment procéder : vous pouvez suivre la démarche suivante :

-1- lisez en entrée l'opération mathématique à réaliser,
-2- convertissez chaque chiffre de l'opération en décimal,
-3- calculez cette opération en décimal,
-4- convertissez le résultat obtenu en chiffres romains,
-5- affichez ce résultat en sortie.

Vous devez faire attention aussi aux cas suivants :

-1- tous les chiffres romains de l'opération sont correctement écrits. Donc vous n'avez pas à vous soucier, par exemple des écritures suivantes : CMM (incorrecte car 1000 après 900), ni CFM (incorrecte à cause de la lettre F) etc.

-2- dans le cas où votre opération mathématique dépasse 3999, vous devez afficher un message d'erreur (voir le fichier in3.txt pour l'entrée et out3.txt pour la sortie obtenue).

-3- il y a au moins un argument sur la ligne de commande, dans le cas contraire un message d'erreur est affiché en sortie (voir le fichier in4.txt pour l'entrée et out4.txt pour la sortie obtenue).

-4- la priorité de traitement dans une opération quelconque entre les opérateurs + et - est de gauche à droite.

Fichiers fournis : Ci-joint 6 jeux d'entrée/sortie vous permettant de tester votre programme.

Fichiers	Type d'opération
in1.txt/out1.txt	Une opération basic
in2.txt/out2.txt	Une opération complexe
in3.txt/out3.txt	Échec, dépassement
in4.txt/out4.txt	Échec, nombre d'arguments
in5.txt/out5.txt	Opération complexe -1-
in6.txt/out6.txt	Opération complexe -2-

En plus de ces 6 exemples, nous pourrions utiliser aussi d'autres exemples, non fournis dans l'énoncé, pour tester votre programme. Ces exemples non fournis vont nous permettre de vérifier que vous avez bien respecté les exigences demandées dans l'énoncé. Ces exemples, s'ils sont utilisés, seront disponibles après la remise des notes du TP#2.

Fichiers à remettre : « exo2tp2h07.cpp » va contenir votre programme. Pour l'exercice -2-, vous devez prévoir deux sortes de rapport : un guide d'utilisation de votre programme qui sera lu par un utilisateur néophyte et un guide pour programmeur lui 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 « tp2.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 tp2 tp2.zip

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

   remise –v ift1166 tp2

Barème : ce TP2 est noté sur 15 points.

Exercice -1-	Explications	5 points
Exercice -2- (10 points)	Compilation et respect des spécifications	1
	Codage, commentaires etc.	3
	Rapport	2.5
	Tests (fournis)	3.5

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

Pour s'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 des 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 exo2tp2h07.exe exo2tp2h07.cpp`
exécution et redirection dans le cas du premier exemple (in1.txt)	`exo2tp2h07.exe < in1.txt > ma_out1.txt`
comparaison dans le cas du premier exemple (ou1.txt)	`diff ou1.txt ma_out1.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 : 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 **tp02**.

Mise à jour

15-05-2007 diffusion

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

Annexe -1-

Comment avoir le nombre d'arguments de la ligne de commande en C++?

La ligne de commande qui lance l'exécution d'un programme peut contenir un ensemble d'arguments. Par exemple, la ligne de commande « **`exo2tp2h07.exe M + M`** » est une ligne de commande qui contient 4 arguments.

Indice de l'argument	Argument	argv[]
0	exo2tp2h07.exe	argv[0]
1	M	argv[1]
2	+	argv[2]
3	M	argv[3]

Dans les langages C et C++, l'argument 0 est le nom du programme (ce n'est pas le cas pour le langage java où le nom du programme n'est pas pris en compte). Il faut déclarer la fonction main de la manière suivante pour pouvoir accéder à ces arguments :

```
int main (int argc, char *argv[]) { //etc.}
```

argc : contient le nombre d'arguments de la ligne de commande.
argv : est un tableau de pointeurs sur les arguments de la ligne de commande.

Pour ce travail pratique, nous allons présenter uniquement l'utilisation de l'argument « argc ». Nous verrons plus en détails l'utilisation de « argv » dans un prochain travail pratique.

Ce simple programme affiche le nombre d'arguments et la liste des arguments de la ligne de commande :

```
#include <iostream>

using namespace std;

int main(int argc, char* argv[]){

    cout << "nbre d'arguments de la ligne de commande est: " << argc << endl;

    for (int i=0;i<argc;i++)
         cout << "argv[" << i << "]: " << argv[i] << endl;

    return 0;

}
```

Annexe -2-

La lecture en C++

Il y a plusieurs manières de lire des données proposées en entrée. Nous avons déjà étudié en cours l'utilisation de « **cout** ». Cette annexe a pour but de vous introduire à d'autres méthodes. Libre à vous de les utiliser ou pas.

-1- int get()

Retourne la valeur du caractère lu, sinon EOF si la fin du fichier est atteinte.

```
#include <iostream>

using namespace std;

int main(){
      char c;
      while ((c=cin.get()) != EOF)
          cout << c;
      cout << "on sort de la boucle!" << endl;
      return 0;
}
```

Le programme lit indéfiniment les caractères à partir du clavier. On arrête la lecture si les touches, « CTRL » et « D » sont appuyées en même temps, sinon lorsqu'on appuie sur la touche retour chariot (return) du clavier. Après l'affichage, le programme se remet en état d'attente de lecture. Pour arrêter complètement le processus et passer à la suite du programme (ici l'affichage de l'expression: on sort de la boucle!), il faudra appuyer sur « CTRL-D » une seconde fois. Un appel du style « CTRL-C » arrêtera complètement le programme sans exécuter la suite des instructions qu'il pourra contenir.

-2- istream& get (char& c)

Extrait le premier caractère du flux, même si c'est un espace, et le place dans le caractère c.

```
#include <iostream>

using namespace std;

int main(){
      char c;
      while (cin.get(c))
          cout << c;
      cout << "on sort de la boucle!" << endl;
      return 0;
}
```

Pour les touches CTRL-D, voir le paragraphe précédent.

-3- istream& get(char* str, streamsize count, char delim)
istream& get(char* str, streamsize count)

Extrait count-1 caractères du flux et place le résultat à l'adresse pointée par ch. La lecture s'arrête au délimiteur delim (première version de la fonction) qui est par défaut le '\n' (la seconde version de la commande) ou la fin du fichier. Le délimiteur n'est pas extrait du flux.

```
#include <iostream>

using namespace std;

int main(){

      char c[80];
      cin.get(c,79);
      cout << c << endl;
      return 0;
}
```

Le caractère '\0' est inséré à la fin de la chaîne. Il faudra s'assurer que le buffer peut contenir `count-1` caractères.

-4- istream& getline(char* str, streamsize count, char delim)
istream& getline(char* str, streamsize count)

La même chose qu'en -3-, sauf qu'ici le délimiteur est extrait du flux, mais il n'est pas recopié dans le tampon. Le délimiteur est extrait du flux (lu), à condition de se trouver dans les `count-1` caractères à lire.

-5- istream& read(char* str, streamsize count)

Cette fonction extrait un bloc de taille `count` et elle le place dans `str`. Il faut s'assurer que `str` est d'une taille suffisante pour contenir ce bloc. Elle est utilisée généralement pour la lecture binaire.

```
#include <iostream>

using namespace std;

int main(){

      char c[] = "??????????";
      // on ne va lire que 4 valeurs.
      cin.read(c,4);
      cout << endl << c << endl;

      return 0;

}
```

En entrée: `1234`

En sortie: `1234??????`

Annexe -3-

Q: Est-ce qu'on doit ouvrir un fichier?
R: Non. Vous n'avez pas besoin d'ouvrir un fichier. Dans le « Tp#1 », vous avez redirigé la sortie vers un fichier « xyz.txt » en utilisant l'opérateur « > ». Est-ce que vous avez ouvert le fichier « xyz.txt » pour écrire les résultats obtenus? Non. C'est le système qui, en voyant l'opérateur « > » a ouvert le fichier pour vous et a écrit les données pour vous. Idem pour l'opérateur « < ». Si vous décidez de l'utiliser, le système va ouvrir pour vous le fichier et va lire pour vous son contenu.

Q: Comment lire alors les données?
R: En utilisant le « cin »!

Q: Comment?
R: C'est juste une lecture. Il vous manque juste à faire une petite analyse très simple pour savoir quoi utiliser avec le « cin ». Il y a plusieurs façons de le faire. Se servir des informations fournies en annexe -2- et/ou -3- ou tout simplement les opérateurs de structures de contrôle.

Q: Je ne vois pas le lien entre l'annexe -2- et le reste de l'énoncé, pourrais-je avoir des explications ?
R: D'après moi, l'annexe -2- n'est pas utile. Mais comme chacun a sa propre façon de coder, je voulais anticiper une question pour ceux/celles qui veulent savoir comment avoir le nombre d'arguments sur la ligne de commande. Idem pour l'annexe 3.

Q: C'est quoi la différence entre : « exo2tp2h07.exe M + M » et « exo2tp2h07.exe < inxyz.txt »
R: Elles sont identiques!
Que vous fassiez « exo2tp2h07.exe M + M » ou bien que vous écriviez dans le fichier « inxyz.txt » l'expression « M + M » puis vous lancez le programme avec la commande « exo2tp2h07.exe < inxyz.txt » ; techniquement parlant les deux opérations sont équivalentes. La 2e opération, elle évite de taper à chaque fois l'opération « M + M »! Vous pouvez vous entraîner avec le petit exercice suivant (fichier test.cpp):

```
#include <iostream>

using namespace std;

int main(){

      int x=0;
      cin >> x;
      cout << x;
      return 0;

}
```

Vous exécutez le programme avec « test.exe 1 ». Puis vous créez le fichier « fic.txt » et vous écrivez à l'intérieur le chiffre « 1 ». Vous exécutez la commande suivante : « test.exe < fic.txt ». Vous allez arriver à la conclusion que les deux façons d'opérer sont équivalentes!

Q: Dans votre énoncé de tp02, il est indiqué dans le barème que 3.5 points des notes est accordés pour les tests. De quels tests s'agit-il?
R: Les tests fournis: les fichiers in/out 1 à 6. Les tests non fournis: d'autres tests avec d'autres opérations mathématiques mais tout en respectant l'énoncé. On ne va pas tester par exemple la multiplication!