IFT1166 – TRAVAIL PRATIQUE #2 – 22 septembre 2005

Minerva, les romains et la calculette ...

Mohamed Lokbani

Équipes : Le travail est à faire en monôme (**une seule personne**).
Remise : Deux remises à effectuer : électronique et papier le **mardi 11 octobre, 20h30 au plus tard, sans possibilité de retard**. **La copie papier doit-être remise à vos démonstrateurs ou pendant le cours du 11 octobre.**
Conseil : N'attendez pas le dernier jour avant la remise pour commencer, vous n'aurez pas le temps.

But : Ce TP a pour but de vous faire pratiquer l'algorithmique de base ainsi que les notions de C++ déjà apprises telles que l'utilisation de fonctions. Il n'est pas donc nécessaire à ce stade d'utiliser des structures ou des classes.

Enoncé : On se propose d'écrire un programme C++ permettant d'effectuer les opérations d'addition et de soustraction sur des chiffres romains.

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 pas plus qu'un seul D, L, ou V dans ce nombre la. 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.

Travail à réaliser : Votre tâche est d'écrire un programme en C++ qui prend en entrée une opération mathématique en chiffres romains et qui donne le résultat en chiffre romain de cette opération. Par exemple:

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

Pour ce faire, vous devez procéder ainsi:

-1- Lire en entrée l'opération mathématique à réaliser,
-2- Convertir chaque chiffre de l'opération en décimal,
-3- Calculer cette opération en décimal,
-4- Convertir le résultat obtenu en chiffres romains,
-5- Afficher ce résultat en sortie.

Vous devez faire attention aussi aux cas suivants:

-1- Tous les chiffres romains de l'opération son correctement écrits. Vous n'avez pas donc 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 afficher 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 4 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

En plus de ces 4 exemples, nous utiliserons 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 respectés les exigences demandées dans l'énoncé. Ils seront disponibles après la remise des notes du TP#2.

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 (qui normalement fonctionne correctement chez vous) fonctionne bien sur les ordinateurs du DESI. Pour avoir la version du compilateur utiliser la commande : "**gcc –v**" devra donnée le numéro de version "**3.2.4**".

Vous devez remettre 2 fichiers : « tp2.cpp » et « rapport ». Le fichier « tp2.cpp » va contenir votre code alors que le fichier « rapport » va décrire comment vous avez procédé pour réaliser ce travail (pour le contenu d'un rapport voir la FAQ sur la page web du cours).

La remise comprend **3** (**TROIS**) choses (**avant le mardi 11 octobre à 20h30**) :

1. Envoyez vos fichiers « tp2.cpp » et le rapport par la procédure de remise électronique habituelle (Pour obtenir de l'aide sur cette commande, tapez dans un xterm : man remise). Respecter les noms des fichiers.

   remise ift1166 tp2 tp2.cpp rapport

2. Vérifier que la remise s'est effectuée correctement.

   remise –v ift1166 tp2

3. Remettez une **copie papier** de votre programme (« tp2.cpp ») ainsi que de votre rapport.

Barème : Ce TP2 est noté sur 10 points.

Compilation et respect des spécifications	**1**
Codage, commentaires etc.	**3**
Rapport	**2.5**
Tests (fournis et non 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.
- La non remise papier vous pénalise de 1 point.
- Les programmes ne contenant pas d'en-tête, -1 point.
- Un programme qui ne compile pas : 0/10.
- Un programme qui compile mais ne fait les choses prévues dans la spécification : 0/10.
- Les avertissements (warnings) non corrigés : cela dépend de la quantité! À partir de -0.25 et plus.
- Le nom respect du nom de fichier, donc forcement il ne va pas compiler et un des points de la spécification n'a pas été respecté : 0/10.
- 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. Pour chaque exemple (**in{1,2,3,4}.txt**), vous devez produire une sortie **identique** au fichier de sortie (**out{1,2,3,4}.txt**) correspondant.

Pour s'assurer de cela, commencer d'abord par rediriger la sortie dans un fichier avec le symbole **>**. Par exemple pour le premier test fourni (int1.txt/out1.txt)

```
tp2.exe < in1.txt > masortie1.txt
```

Comparer 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 », ajuster juste les options) comme suit :

```
diff masortie1.txt 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

22-09-2005 diffusion

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 « tp2.exe M + M » est une ligne de commande qui contient 4 arguments.

Indice de l'argument	Argument	argv[]
0	tp2.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 entée. 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??????`