IFT1169 – TRAVAIL PRATIQUE #2 – 14 février 2007

« On met la seconde sous un tapis surchargé de neige. »

Mohamed Lokbani

Équipes: le travail peut-être fait en binôme. Vous ne remettez alors qu'un travail par équipe.
Remise : une seule remise à effectuer par la voie électronique le jeudi **15 mars 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 la manipulation de fichiers binaires ainsi que la surcharge des opérateurs.

Exercice 1 : écrire un programme qui permet de faire ce qui suit:
- génère aléatoirement un entier N positif non nul.
- génère aléatoirement N entiers compris entre -1000 et +1000 (inclusivement)
- calcule les statistiques suivantes: le nombre d'entiers positifs, négatifs, pairs et impairs.

La lecture, la sauvegarde et l'affichage des données doivent avoir lieu comme suit :

- si l'option « -s » est présente, les données générées doivent être stockées dans un fichier binaire. L'option « -o » permet de préciser le nom du fichier de sortie sinon l'affichage est renvoyé sur la sortie standard (on pourra utiliser par exemple la redirection pour préserver le résultat dans un fichier). On ne fera pas de calcul statistique à ce stade. On n'écrit dans le fichier que les données générées et on renvoie sur la sortie standard des informations utiles sur le processus de génération et de sauvegarde des données.
- si l'option « -l » est présente, les données seront lues à partir d'un fichier binaire passé en argument à l'option « -o ». L'option « -o » permet donc de préciser le nom du fichier en entrée sinon la lecture se fera de l'entrée standard (on pourra utiliser par exemple la redirection pour lister un fichier de données). Le calcul statistique doit être effectué dans cette étape. Le programme affichera en sortie les résultats du calcul de manière la plus informative que possible sans que le format d'affichage ne soit trop envahissant (trop d'informations tuent l'information!).
- la fonction « main » doit être définie dans le fichier « tp2exo1.cpp ».

Exercice 2 : écrire la définition des classes et le programme permettant de les utiliser en tenant compte de ce qui suit :

- la classe « Point » où un point est représenté par ses coordonnés « x » et « y » dans l'espace plan.
- ajustez la classe « Point » pour permettre d'afficher directement en sortie les coordonnées « x, y » d'une instance de « Point » (i.e. permettre cette instruction : [cout << P; // où « P » une instance de « Point »]).

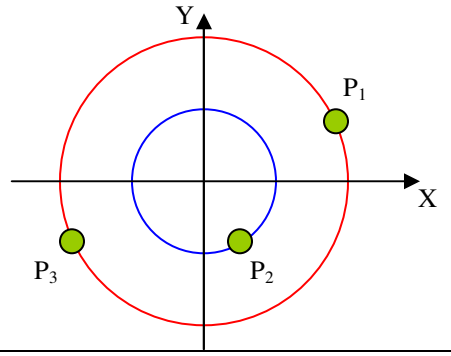
Définition : si tous les points ont la même origine « x_o,y_o », on dit qu'un point « p_1 » est inférieur à un point « p_2 » si le point « p_1 » est plus proche de cette origine que le point « p_2 ». Les points « p_1 » et « p_2 » seront considérés « identiques » si cette distance est la même pour les deux. Dans ce cas, ils sont situés sur le même cercle. La distance d'un point par rapport à l'origine est calculée ici par l'équation $[x^2+y^2]$[1].

- en tenant compte de la précédente définition, complétez la classe « Point » avec les opérateurs « = = », « != », « < », « <= », « > », « >= ».

- écrivez la définition de la classe « Polygone » caractérisée par son nombre de sommets « N » et par ses sommets, stockés dans le tableau « *tab ». On utilise pour cela la classe « Point » définie précédemment.

[1] Dans cet exemple il n'est pas nécessaire de calculer la racine carrée de la distance entre le point et l'origine. Par ailleurs on peut supposer que le point d'origine se situe à (0,0).

- ajustez la classe « Polygone » pour permettre d'afficher directement sur la sortie standard les coordonnées (x, y) des différents sommets d'une instance de « Polygone » : [cout << Pg; // où Pg une instance de Polygone].
- surchargez l'opérateur d'indexation « [ ] » qui retourne le « ième » « Point » du polygone. Cet opérateur lèvera une exception du type entier qui renverra l'indice demandé quand celui-ci n'est pas valide (i.e. est négatif ou supérieur au nombre de sommets).
- l'en-tête de la classe « Point » doit être défini dans le fichier « Point.h » et la définition de ses méthodes dans le fichier « Point.cpp ».
- l'en-tête de la classe « Polygone » doit être défini dans le fichier « Polygone.h » et la définition de ses méthodes dans le fichier « Polygone.cpp ».
- écrivez un programme pour tester les différentes fonctionnalités des classes « Point » et « Polygone ». La fonction « main » doit être définie dans le fichier « tp2exo2.cpp ».



P_1 est identique à P_3 i.e. l'expression « P_1 == P_3 » doit retourner « vraie ». Les deux points sont sur le même cercle. La distance de P_1 à l'origine est la même que celle de P_3 à l'origine.

P_1 est différent de P_2 i.e. l'expression « P_1 != P_2 » doit retourner « vraie ». Les deux points sont sur des cercles différents. La distance de P_1 à l'origine est plus grande que celle de P_2 à l'origine.

Hypothèses et contraintes :

- pour commencer, vous devez déjà respecter les différentes contraintes de programmation précédemment décrites.
- c'est un travail à faire **SEUL** ! Et seul ne signifie pas deux ni dix. Seul signifie aussi **réfléchir seul**. Plagiat équivaudrait à un 0 pour commencer.
- IFT1169 est un cours avancé 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 compliquer le travail pour rien!
- vous vous assurez de ne pas changer les noms des fichiers. Le format de l'affichage en sortie est libre.

Rapport : le fichier « rapport.pdf » 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). 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 aux formats « pdf » ou « ps » (voir la aussi la FAQ sur la page web du cours : comment générer un « pdf »).

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 doit normalement fonctionner correctement chez vous) fonctionne bien aussi 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).
Vous devez d'abord regrouper l'ensemble des fichiers dans un seul fichier « tp2.zip ». Par la suite vous devez faire la remise comme suit :

1. Connectez-vous à « https://remous.iro.umontreal.ca/client » et envoyez le fichier « tp2.zip » via la page web du « tp2 ».

2. Envoyez-nous par email à l'adresse « dift1169@iro.umontreal.ca » avec comme sujet « remise tp#2 », une copie du fichier « tp2.zip ».

Barème : chaque exercice de ce TP2 est noté sur 6 points pour un total de 12 points. Le détail pour chaque exercice est comme suit :

Compilation, respect des spécifications, codage et commentaires	**2**
Rapport	**2**
Tests	**2**

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 : nous allons utiliser des scriptes pour compiler et exécuter votre programme. Assurez-vous de respecter les noms du fichier précédemment mentionnés, ainsi que les noms des différentes options.

Ainsi donc, au moment de la remise de votre travail, vous avez déjà une idée approximative de la note que vous allez obtenir.

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 **tp02**.

Mise à jour

14-02-2006 diffusion de l'énoncé.

Annexe

Pour pouvoir utiliser les fonctions permettant de générer des nombres aléatoires, on a besoin d'inclure dans le programme les fichiers d'entête « cstdlib » et « ctime ».

L'instruction « srand(time(NULL)) ; » permet d'initialiser le générateur pseudo aléatoire pour éviter qu'il ne retourne les mêmes valeurs après le même appel au programme.

L'instruction « int valeur = rand() ; » retourne dans la variable « valeur » un nombre aléatoire entier compris entre « 0 » et « RAND_MAX » inclusivement. « RAND_MAX » est une constante définie dans le fichier d'entête « cstdlib ».

L'instruction « int valeur = (rand()%100) ; » retourne dans la variable « valeur » un nombre aléatoire entier compris entre « 0 » et « 99 ».

L'instruction « int valeur = (rand()%100 + 1) ; » retourne dans la variable « valeur » un nombre aléatoire entier compris entre « 1 » et « 100 ».