IFT339 : STRUCTURES DE DONNÉES (T.P. 2)

DANIEL LEMIRE, PH.D.

1. Introduction

Un des objectifs du cours IFT339 est de faire en sorte que l'on puisse programmer plus facilement et plus rapidement des applications solides. On pourrait croire que d'étudier des théories abstraites comme les structures de données ne nous aide pas concrètement, mais ce deuxième t.p. vise à prouver le contraire. Dans le premier t.p., nous avons résolu un problème simple en utilisant une approche difficile (les piles, les files...). Nous allons maintenant faire l'inverse, soit résoudre un problème relativement difficile en toute simplicité. L'objectif de ce t.p. est de renforcer votre compréhension des flots et de la méthodologie orientée-objet tout en continuant d'utiliser des structures de données importantes. Une solide compréhension des gabarits (*templates*) est encouragée par ce t.p.

2. Théorie

On dit qu'un ensemble S est *ordonné* s'il existe une relation $\leq$: $S \times S \rightarrow \{vrai, faux\}$ telle que

- $a \leq a \forall a \in S$
- $a \leq b$ et $b \leq c$ alors $a \leq c$
- si $a \leq b$ et $b \leq a$ alors $a = b$

et on dit que S est *totalement ordonné* si tout sous-ensemble non-vide de S contient un minimum (élément plus petit ou égal à tous les autres). On se rappelera aussi qu'un ensemble ne peut pas contenir deux fois le même élément.

On a vu dans le cours que l'on devait souvent définir notre propre relation de d'ordre entre des objets. Il est généralement préférable de s'assurer que notre relation d'ordre va créer un ensemble ordonné. (Pourquoi ?)

3. Énoncé

Vous devez créer une classe Individu qui puisse contenir le nom, le prénom, le nom de la ville, l'âge, et la province d'un individu (on peut supposer qu'il n'y a pas d'espaces dans les noms). Votre classe doit être doté d'un constructeur par défaut et des opérateurs "$<$" et "$==$". Pour les fins de ce travail, on dira qu'un individu est "inférieur" à un autre, s'il est plus jeune. On dira qu'ils sont égaux si et seulement s'ils contiennent exactement les mêmes informations (nom, prénom, etc.). Vous devez aussi surcharger les opérateurs $>>$ et $<<$ pour que l'on puisse lire et écrire des "Individu" dans un istream et ostream respectivement selon le format "nom prenom nomdelaville province age". Vous pouvez supposer que chaque chaîne de caractères ne dépasse pas 255 caractères (on ne travaille pas pour la NASA).

Date: 26 mai 2001 (à remettre le 13 juin 2001 à minuit).

Vous devez lire un nombre inconnu d'"Individus" dans un fichier, les mettre dans un liste et les trier pour ensuite les écrire dans un autre fichier (triés). La syntaxe doit prendre la forme "tri fichierentrée.txt fichiersortie.txt".

3.1. **Directives concernant le STL.** Il est suggéré d'utiliser la classe "list" de la librairie STL. Cela n'est pas obligatoire, mais dans tous les cas, **vous devrez utiliser une structure de donnée appropriée et discutée dans le cours** . Nous avons vu en classe des rudiments de STL. Sans exiger ici que vous connaissiez bien cette librairie importante, nous allons néanmoins vous proposer de l'utiliser pour se simplifier la vie. Il est important de préciser qu'il n'est pas nécessaire de connaître ou d'étudier STL pour réussir ce t.p. et que les informations pertinentes seront données librement en classe en ce qui concerne STL. N'hésitez pas à poser des questions ! Pour utiliser la classe "list", il suffit de commencer votre code par "#include <list>" suivi de "using namespace std ;". Par la suite, si vous voulez créer une classe "list" qui puisse contenir des "Individus", vous pouvez le faire avec la syntaxe "list<Individu> maliste ;" ce qui a pour effet de créer une liste vide. Vous pouvez ensuite utiliser les fonctions-membres suivantes (VOUS N'AVEZ PAS BESOIN DE TOUT ÇA !) :

- maliste.push_back(ind) ; // ajoute l'objet "ind" à la fin de la liste, donne "void"
- maliste.pop_back() ; // retire l'objet de la fin de la liste, donne "void"
- maliste.push_front(ind) ;// ajoute l'objet "ind" de la fin de la liste, donne "void"
- maliste.pop_front() ;// retire l'objet du début de la liste, donne "void"
- maliste.sort() ; // effectue un tri sur la liste (note : on peut passer une fonction-objet en paramètre, ex. sort(greater<Individu>())... attention cependant aux opérateurs que vous avez définis !)
- maliste.size() ; //donne le nombre d'éléments dans la liste
- maliste.clear() ; // met à zéro la liste, donne "void"
- maliste.empty() ; // est-ce que la vide est vide ? donne un bool
- maliste.begin() ; // donne un itérateur bidirectionnel de classe "list<Individu> : :iterator" pointant sur le début de la liste
- maliste.end() ; // donne un itérateur bidirectionnel de classe "list<Individu> : :iterator" pointant sur la fin de la liste (+1)
- maliste.erase(it) ;// supprime l'élément pointé par l'itérateur (donne un itérateur pointant après)
- maliste.insert(it, ind) ;// insère un élément après la position indiquée par l'itérateur (donne un itérateur pointant sur le nouvel élément)

On se rappelle qu'un itérateur bidirectionnel peut être incrémenté (it++) et décrémenté (it–) à volonté. On peut traverser la liste en faisant "list<Individu> : :iterator p = li.begin() ; while(p != li.end()) { cout << *p << endl ;}"

4. À Remettre

Le travail sera évalué sur 100 points. Vous devez remettre un rapport qui répond aux questions suivantes :

(1) Donner un exemple illustrant la nécessité de ne pas choisir une relation d'ordre arbitrairement. Indice : définissez un objet avec une surcharge de l'opération $\leq$telle que l'algorithme de tri-fusion ou le quicksort (au choix) donne un résultat aberrant. Expliquez ensuite pourquoi le résultat aberrant

obtenu avec votre exemple ne serait pas possible si votre relation menait à un ensemble ordonné. (10 points)

(2) Si on traite des chaînes de caractères sous la forme de pointeurs (char*), pourquoi est-il important de faire attention à l'opération d'égalité "==", donnez un exemple. Est-ce que vous pouvez proposer une solution ? (10 points)

(3) Si on considère pour les fins de cette question que $a \leq b \Leftrightarrow (a < b)|(a == b)$, est-ce que la relation d'ordre qu'on vous demande d'utiliser pour ce t.p. donne un ensemble ordonné ? Expliquez. (10 points)

(4) Le tri se fait uniquement sur l'âge. On a vu qu'on pouvait passer en paramètre une fonction-objet aux méthodes du STL pour modifier dynamiquement la relation "<" qui est utilisée par défaut. Pourquoi est-ce que je ne pourrais obtenir le même résultat par héritage. (Aucun point accordé pour cette question, mais essayez d'y répondre pour la forme !)

(5) Quel est l'avantage d'utiliser une structure de donnée appropriée dans le cas qui nous concerne ? Si vous avez choisi d'utiliser STL, expliquez le gain que vous en retirer. Si vous n'aviez pas pris le cours IFT339, est-ce que vous auriez pu faire ce problème aussi facilement, aussi efficacement ? (Aucun point n'est accordé pour cette question, mais vous devriez y réfléchir !)

Vous devez aussi remettre les sources qui répondent à l'énoncé plus haut électroniquement et sur papier. D'autres instructions seront données en classe. Vous devez vous assurer que vos sources compilent sous GCC pour les Sun du département (20 points) et que le tri s'effectue correctement. (50 points) Il est permis de travailler en équipe de deux (et de ne remettre qu'un seul rapport par équipe). On peut normalement faire le problème en utilisant très peu de code...