Final

Mercredi 23 Juin 2010, de 10h30 à 12h30

Note : Documents cours et TD papier autorisés. La copie du voisin même papier, ne l'est pas.

1. Produit scalaire. (6 pts)

Supposons que l'on travaille avec des vecteurs dont la taille N est connue à la compilation. Nous souhaitons que le produit scalaire de vecteurs soit partiellement déroulé pendant la compilation afin d'éliminer les délais de branchement qu'imposerait une boucle pendant l'exécution

Ce que nous voudrions c'est qu'une suite d'instructions comme :

```
Vecteur a[N], b[N];
double s = dot(a, b);
```

soit transformée en :

```
Vecteur a[N], b[N];            // pour simplifier Vecteur sera équivalent à double
double s = a[0]*b[0] + a[1]*b[1] + ... + a[N-1]*b[N-1];
```

où N désigne la longueur des vecteurs, constante connue à la compilation.

- Codez la fonction dot sous forme de méta-programme.

(Bien sûr il est possible de dérouler la boucle à la main, mais c'est ce que nous voulons éviter).

- Modifiez le code pour éviter le calcul d'indexation sur des tableaux et obtenir un code équivalent à :

```
double a[N], b[N];  // N taille des tableaux
double *aa=a, *bb=b;
double s = (*aa++) * (*bb++)  + (*aa++) * (*bb++)  + ... + (*aa++) * (*bb++);  * N fois
```

2. Manipulation de séquence de liste (4 pts)

Vous disposez de deux listes de types (vues en cours). Pour des besoins de compilation, nous souhaitons supprimer de l'une des listes, tous les éléments de la seconde. Ecrivez un outil permettant de réaliser ceci.

Rappel de cours :
Une liste de types repose sur l'utilisation récursive d'un template et un type représentant la fin de liste

```
template <class T, class U>
struct type_list {  typedef T Tete;
                    typedef U Queue ;
};
```

```
class Vide {} ;
```

```
typedef type_list < char,
                    type_list < unsigned char,
                                type_list < signed char,
                                            type_list < wchar_t, Vide> > > > > types_caracteres ;
```

Tete

Queue

Fin de liste

3. Exploitation de la STL (10 pts)

Nous disposons d'un vecteur d'objets Eleve.

```
Eleve{
     nom, prenom ;
     adresse ;
    Classe  *ptcla ;
    …
    *ptResultat ;
}
```

Nous souhaitons déterminer la liste des 10 meilleurs d'une classe afin de les féliciter pour leurs brillantes prestations.

3.1. Implantation de base

Ecrivez une fonction *void* ***selectionClasse****(vector<Eleve *> &vr, vector<Eleve> &vs, Classe &cla)* permettant de créer un vecteur *vr* en sélectionnant au sein du vecteur source *vs*, les élèves d'une classe dont la référence est fournie en paramètre.
Ecrivez une fonction *float* ***moyenneEleve****(**const** Eleve &e)* permettant d'obtenir la moyenne d'un élève.
Ecrivez une fonction *vector<Eleve *> &***selectionEleve****(vector<Eleve *> &vs)* permettant d'obtenir en début de vecteur les 10 meilleurs. Vous pouvez choisir de seulement les regrouper en tête ou de les obtenir triés selon un ordre décroissant de moyenne. Si vous avez besoin de créer d'autres fonctions ou classes, précisez les objectifs de ces créations et la façon de les utiliser.

3.2. Implantation avec les algorithmes

Le professeur ne sait vraiment pas quoi faire ! Pour embêter, il a décidé que le résultat doit être obtenu **sans écrire de boucle for ou autres, ni récursivité**, mais en utilisant encore plus que précédemment les algorithmes de la STL.

Ecrivez une fonction ***void convertir****(vector<Eleve *> &vr, vector<Eleve> &vs)* qui permet de dimensionner et remplir un vecteur *vr* de pointeur transmis en paramètre, sur la base d'un vecteur d'objet Eleve *vs*. Pour ce faire vous commencerez par écrire une fonction *Eleve *****to_pt****(Eleve &e)* que vous utiliserez explicitement comme foncteur pour transformer le vecteur d'élèves *vs* en vecteur de pointeur vr.

Ecrivez une fonction ***void selectionClasse2****(vector<Eleve *> &vr, vector<Eleve *> &vs, Classe *cla)* qui permet de ne copier dans le vecteur *vr* que les pointeurs de *vs*, des étudiants appartenant à la classe désignée par le pointeur transmis en paramètre. Pour ce faire vous commencerez par écrire une fonction ***bool predSelect****(**const** Eleve *e1, **const** Classe *cla)* qui retourne un booléen précisant que l'étudiant n'est pas dans la classe. Vous utiliserez cette fonction explicitement comme foncteur pour rejeter de la copie les pointeurs d'élève non souhaités. Attention, cette fonction nécessite deux paramètres et le second sera toujours le même !

Ecrivez une fonction **float moyenneEleve2**(**const** Eleve &e) procédant à la somme des notes d'un élève pour en restituer la moyenne.

A ce stade la fonction de détermination des 10 meilleurs ne doit pas avoir besoin d'être modifiée.