Algorithmique et structures de données 2

X4I0030 **2015/2016**

Travaux pratiques

Rappels, Structures de données linéaires, Objets (3 séances)

Tous Ensembles

1) Écrire en C++ une classe `EnsembleC` permettant de manipuler un ensemble de chaînes de caractères (de 26 caractères au maximum) avec les méthodes décrites ci-dessous et séparément un programme en utilisant des instances. Le programme doit demander à l'utilisateur un texte et stocker dans un premier ensemble les mots commençant par la lettre « a », dans un autre ensemble les mots ayant un « b » en seconde position, dans un troisième les mots ayant « c » comme troisième lettre, et ainsi de suite jusque « z ». Ces différents ensembles seront regroupés dans un `Vector` d'`EnsembleC`. Le programme devra, une fois la saisie terminée, afficher les mots stockés ayant « de » comme 4ème et 5ème caractères.

   La classe `EnsembleC` doit être implémentée sous forme d'un chaînage simple, les maillons étant une simple structure privée. Chaque mot ne doit figurer qu'une fois dans l'ensemble. L'ajout d'un nouveau mot se fait en queue de chaînage, le nombre de maillons est stocké. Chaque méthode doit être commentée avec ses PRE- et POST-condition et les formes des données au mieux et au pire avec les ordres de grandeur de coût temporel associés.

```
EnsembleC()          // constructeur, initialise un ensemble vide, appelé automatiquement par « new »
~EnsembleC()         // destructeur, libère la mémoire
boolean estVide()    // vrai ssi l'ensemble est vide
boolean contient(Chaîne mot) // vrai ssi mot est un élément de l'ensemble
void ajoute(Chaîne mot)    // ajoute un mot à l'ensemble, ne fait rien s'il y est déjà
void retire(Chaîne mot)    // supprime le mot de l'ensemble, ne fait rien s'il n'y est pas
chaîne contenu()   // concatène les mots stockés, séparés par un unique espace
```

Exemple de déclaration et d'utilisation d'une classe en C++ (les fichiers complets sont sur Madoc, dossier et archive Point) :

fichier point.hpp

```
// Définition du type point en coordonnées
// cartésiennes
#include <string> // pour le type std::string

/***********************************************/
class Point
{
   private :
      float _x,_y; // coordonnées cartésiennes

   public :
      /* création/destruction */
      Point(); // constructeur par défaut --> origine
      ~Point(); // destructeur

      std::string enChaine() const;
                  // représentation sous forme "(x,y)"

      void deplacer(float dx, float dy);// translation de (dx,dy)

      void tourner(float phi);              // rotation de phi°
}; // class Point
```

fichier point.cpp

```
// Définition des méthodes de Point
#include "point.hpp"
#include <cmath>        // pour les fonctions mathématiques
#include <sstream>      // pour les conversions en chaine

using namespace std; // seulement dans le .cpp !
////////////////////////////////////////////////
Point::Point()
{  _x = 0.0;
   _y = 0.0;
}
////////////////////////////////////////////////
Point::~Point() {} // rien à faire
////////////////////////////////////////////////
string Point::enChaine() const
{  stringstream sst; // pour la création du résultat
   sst << "(" << _x << "," << _y << ")";
   return sst.str();
}
////////////////////////////////////////////////
void Point::deplacer(float dx, float dy) { … }
////////////////////////////////////////////////
void Point::tourner(float phi) { … }
```

fichier testPoint.cpp

```
// Programme utilisant la bibliothèque point
#include <iostream> // utile pour les entrées/sorties
#include "point.hpp" // pour utiliser le type Point
using namespace std;

int main() // fonction principale
{  Point p; // point qui sera déplacé
     …
   cout << "Point actuel : " << p.enChaine() << endl;
     …    p.deplacer(deltax,deltay);
     …    p.tourner(angle);
   return 0;
} // main
```

La compilation se fait séparément :

- `gpp point.cpp -c`

  permet de créer le fichier point.o ; inutile de recompiler point.cpp tant qu'il n'a pas été modifié

- `gpp testPoint.cpp point.o -o testP.exe`

  compile le programme ; une modification de `testPoint.cpp` ne nécessite que cette ligne

(ou utiliser le makefile fourni : `make all`)

2) Écrire ensuite en C++ une version générique de la classe `Ensemble` avec des méthodes similaires et réécrire le programme principal pour l'utiliser. La classe `EnsembleC` doit se comporter comme la classe `Ensemble<T>` lorsque `T` désigne le type `string`.

   La fonction `contenu` utilisera un `stringstream` en supposant (sans le vérifier) que le type `T` répond à l'opérateur « << ».

   Vous trouverez sur Madoc un exemple de déclaration et d'utilisation d'une classe paramétrée en C++ (les points du plan) dans le dossier (et l'archive) `pointGenerique`. Attention, la classe `Point` étant générique, il n'est cette fois pas possible de la compiler séparément (avec l'option -c), son code est inclus par transitivité, donc `gpp testPoint.cpp -o testPoint.exe` suffit.