Quelques exercices autour des objets

Christophe Labourdette

Octobre 2016

1 On veut construire un type Fraction

1.1 Définition de la classe

Définir une classe Fraction avec deux champs privés correspondant au numérateur et au dénominateur, et :

1. un constructeur sans argument (0/1),
2. un constructeur à deux arguments (n/d),
3. un constructeur à un argument (n/1),
4. une fonction membre int numerateur() et une fonction membre int denominateur(),
5. une fonction membre int signe().

1.2 Surcharge d'opérateurs

Surcharger les opérateurs : *, +, - (les deux !), /, «.

1.3 Autres méthodes

Définir une fonction membre void Reduction() pour simplifier les fractions. Ecrire une fonction membre float() qui retourne le flottant associé à une fraction. Tester les problèmes de conversion.

2 Géométrie

1. On définit une classe Point qui servira à représenter informatiquement des points dans l'espace R (où n est quelconque et peut être différent entre les différents points).
   Chaque objet, instance de la classe Point contiendra les attributs suivants :
   — le nom du point,
   — la dimension n de l'espace dans lequel vit le point,
   — l'ensemble des valeurs des coordonnées du point.
   Pour représenter le nom d'un point, on utilisera la classe standard **std : :string** (ajouter la ligne *#include <string>* au début du fichier). On définira, de plus, les fonctions membres :
   — un constructeur qui prend un entier n positif en paramètre et initialise un point dans R dont toutes les coordonnées sont nulles,
   — un destructeur,
   — une fonction setX qui prend en paramètres l'entier i et une valeur réelle et permet de donner une valeur à la ième coordonnée,
   — une fonction getX qui prend en paramètre un entier i et permet de récupérer la valeur de la ième coordonnée.
   Les deux dernières fonctions membre doivent contrôler la validité du paramètre entier. Dans toutes les fonctions membre ci-dessus, ajouter une instruction qui affiche un message pour suivre les différents appels à ces fonctions. Mettre l'interface dans le fichier Point.hpp et l'implémentation dans le fichier Point.cpp.
2. Tester cette classe, dans le programme C++ suivant :

```
#include "Point.hpp"
#include <iostream>
int main ( ) {
        // cree deux points
        Point p(5) , q(10) ;
        // initialise les coordonnees
        int i ;
        for (i=0; i<5; i++) p.setX(i ,3.5*i);
        for (i=0; i<10; i++) q.setX(i, 2*i) ;
        // petit calcul
        for (i=0; i<5; i++)
                p.setX(i ,p.getX(i) + q.getX(2*i) ) ;
        //affichage d'un des points
      for (i=0; i<5; i++)
                std::cout<< i << "␣" << p.getX(i) << std::endl;
}
```

Compiler et tester le programme.
Vérifier en particulier l'appel des différentes fonctions membre de la classe Point (constructeurs, destructeur, etc).

3. Ajouter la ligne Point r(p) ; entre les lignes 14 et 15.
   Compiler et tester le programme.
   Définir un constructeur par recopie (dans lequel, on affichera un message pour tester si ce constructeur est utilisé).
   Compiler et tester le programme.
4. Ajouter la ligne r = q ; après la précédente.
   Compiler et tester le programme.
   Définir un opérateur d'assignation (dans lequel, on affichera un message pour tester si cet opérateur est utilisé.
   Compiler et tester le programme.
5. Ecrire une fonction C++ (en dehors de la classe Point) qui prend en argument 2 objets de type Point et retourne comme résultat la distance (euclidienne) de ces deux points.
   Remarque
   Si on fournit deux points ayant un nombre différent de coordonnées, on supposera que les "coordonnées manquantes" sont nulles.

3 Pointeurs et liste chaînée

L'objectif est de manipuler des listes chaînées d'entiers. On considère la classe Element :

```
struct Element {
  int val;
  Element * suivant;
};
```

Le champ suivant contient l'adresse du prochain bloc element (on utilise l'adresse 0 pour signifier qu'il n'y a pas d'autre bloc).

1. Ecrire une procédure Element * Ajouter(int v, Element * l) qui ajoute un élément de valeur v dont le successeur pointe sur l et retourne l'adresse de ce nouvel élément.
2. Ecrire une fonction int Longueur(Element * l) qui compte le nombre de bloc element à partir de l'adresse l.
3. Ecrire une procédure void Affiche(Element * l) qui affiche tous les valeurs entières contenues dans la liste commençant à l'adresse l.
4. Ecrire une fonction Element * Copier(Element * l) qui renvoie une copie de la liste l
5. Ecrire une fonction Element * Inverser(Element * l) qui renvoie une la liste l inversée.
6. Ecrire une procédure void Detruire(Element * l) qui désalloue tous les blocs de la liste l.
7. Ecrire une fonction Element * AjouterF(int v, Element * l) qui ajoute un élément de valeur v à la fin de la liste l, et qui retourne l'adresse du premier bloc de la liste.