Université de Nice-Sophia Antipolis — POLYTECH
ELEC4 — 2017–2018

Examen de Programmation C++

Durée : 1h
Aucun document autorisé
Mobiles interdits — Nom : — Prénom :

Afin de gagner de la place mémoire, on définit des matrices de réels dites creuses qui ne mémorisent que des valeurs différentes de 0. Une matrice creuse $M \times N$ sera représentée par :

- un nombre de lignes ;
- un nombre de colonnes ;
- une suite de triplets. Chaque triplet contient l'indice l de la ligne, l'indice c de la colonne et la valeur réelle double (non nulle) de la case (l, c) de la matrice creuse.

Par exemple, la matrice creuse 3×4 suivante : $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6.5 \\ -3.2 & 0 & 0 & 0 \end{bmatrix}$

est représentée par $\begin{cases} 3 \\ 4 \\ (2, 0, -3.2)(1, 3, 6.5) \end{cases}$

▶ 1. Écrivez la classe `triplet` pour représenter un triplet comme défini précédemment. Vous définirez son constructeur, ses accesseurs et ses mutateurs.

```
#ifndef TRIPLET_HPP
#define TRIPLET_HPP

#include <cassert>

/*
 * La classe triplet représente la valeur d'une matrice creuse de
 * réels.  Elle contient la valeur du réel et ses coordonnées (lignes,
 * colonnes) dans la matrice.
 *
 * @author Vincent Granet (vg@unice.fr)
 */
class triplet {

private:
  int l, c; //
  double valeur;

public:
```



```
  // constructeur
  triplet(const int i, const int j, const double v) :
          l(i), c(j) , valeur(v)  { assert(v=!0); }
  // accesseurs
  double getValeur() const { return this->valeur; }
  int getLigne()     const { return this->l; }
  int getColonne()   const { return this->c; }
  // mutateurs
  void setLigne(const int l)   { this->l = l ; }
  void setColonne(const int c) { this->c = c; }
  /* Antécédent : v!=0 */
  void setValeur(const double v)  {
    assert(v!=0);
    this->valeur = v;
  }
};

#endif
```

On considère que la classe générique `noeud<T>`, donnée en annexe, est à votre disposition pour représenter la suite de triplets.

▶ 2. Écrivez la classe `MatriceCreuse` avec :

1. trois variables membres privées `nbLignes` et `nbColonnes` de type **int** et `lesElements`, une liste linéaire de nœuds qui contient les triplets ;
2. un constructeur à deux paramètres pour initialiser les 3 variables membres ;
3. la fonction `getValeur` qui renvoie le réel (**double**) de coordonnées (l, c) dans la matrice creuse courante. l et c sont paramètres de la fonction.
4. la surcharge de l'opérateur `<<` pour écrire dans un `ostream` la matrice creuse courante. Par exemple, la matrice creuse précédente sera écrite :
   0 0 0 0
   0 0 0 6.5
   −3.2 0 0 0

```
#ifndef MATRICECREUSE_HPP
#define MATRICECREUSE_HPP

#include <cassert>
#include <iostream>
#include "noeud.hpp"
#include "triplet.hpp"

/*
 * La classe MatriceCreuse représente des matrices creuses, i.e. des
 * matrices dont la majorité des valeurs sont nulles. Seules les
 * valeurs non nulles son mémorisées.
 *
 * @author Vincent Granet (vg@unice.fr)
 */
```

```
class MatriceCreuse {
protected:
  int nbLignes;
  int nbColonnes;
  noeud<triplet> *lesElements;
public:
  // constructeur
  MatriceCreuse(const int l, const int c) :
    nbLignes(l), nbColonnes(c), lesElements(nullptr)
  {
    assert(l>0 && c>0);
  }

  // antécédent : l>=0 et l<nbLignes et c>=0 et c<nbColonnes
  // rôle : renvoie le réel de coordonnées (l,c) dans la matrice courante
  const double getValeur(const int l, const int c) const;

  // surcharge de l'opérateur de sortie « pour écrire la matrice
  // creuse p sur l'ostream f
  friend std::ostream&operator<<(std::ostream &f, const MatriceCreuse &m);
};

#endif

#include <cassert>
#include "MatriceCreuse.hpp"

// Antécédent : l>=0 et l<nbLignes et c>=0 et c<nbColonnes
// Rôle : renvoie le réel de coordonnées (l,c) dans la matrice courante
const double MatriceCreuse::getValeur(const int l, const int c) const {
  assert(l>=0 && l<nbLignes &&  c>=0 && c<nbColonnes);
  // chercher dans la liste l'élément de coordonnées (l,c)
  noeud<triplet> *p = lesElements;
  while (p != nullptr) {
    triplet t = p->getElement();
    if (t.getLigne() == l && t.getColonne() == c)
      // trouvé !
      return t.getValeur();
    // else passer au triplet suivant
    p = p->getSuivant();
  }
  // élément non trouvé
  return 0.0;
}

// surcharge de l'opérateur de sortie « pour écrire la matrice
// creuse p sur l'ostream f
std::ostream&operator<<(std::ostream &f, const MatriceCreuse &m) {
  for (int i=0; i<m.nbLignes; i++) {
    for (int j=0; j<m.nbColonnes; j++)
      f << m.getValeur(i,j) << " ";
    //
    f << std::endl;
  }
```



```
  return f;
}
```

.....

Annexe

```
#ifndef NOEUD_HPP
#define NOEUD_HPP

/*
 * la classe générique noeud représente un maillon d'une liste chaînée
 *
 * @author Vincent Granet (vg@unice.fr)
 */

template <typename T>
class noeud {

private:
  T elt;
  noeud<T> *suivant;

public:
  // constructeur
  noeud(T e, noeud<T> *s = nullptr) : elt(e), suivant(s) {}

  // accesseurs
  const T & getElement() const {
    return this->elt;
  }

  noeud<T> *getSuivant() const {
    return this->suivant;
  }
  // mutateurs
  void setSuivant(noeud<T> *s) {
    this->suivant = s;
  }

  void setElement(const T &x) {
    this->elt = x;
  }
};

#endif
```