```
/*
-------------------------------------------------------------------------------
Nom du fichier : matrice.cpp
Nom du labo    : Labo 7 : Vecteur et Matrice
Auteur(s)      : Tim Ernst et Antonio Pollino
Date creation  : 12.12.2021
Description    : Protocole de test des fonctions de la bibliothèques matrice.cpp
Remarque(s)    : -
Compilateur    : Mingw-w64 g++ 11.2.0
-------------------------------------------------------------------------------
*/

#include <cstdlib>
#include <iostream>
#include <limits>
#include "matrice.h"

#define VIDER_BUFFER cin.ignore(numeric_limits<streamsize>::max(), '\n')

using namespace std;

int main(){
   Vecteur v = {1,2,3,4,5,6,7,8,9};

   vector<Matrice> vectorTest;
   vectorTest.push_back({{}});

   vectorTest.push_back({{},
                         {},
                         {},
                         {}});
   vectorTest.push_back({{},
                         {4},
                         {2,-3,1,7,5},
                         {9,8,0}});

   vectorTest.push_back({{},
                         {4,5,6},
                         {2,-3,1},
                         {9,8,0}});

   vectorTest.push_back({{1,2,3},
                         {7,8,9},
                         {2,3,1}});

   cout << boolalpha;
   cout << "Test de la bibliotheque matrice.cpp"  << endl;
   for (Matrice m : vectorTest) {
      cout << "---------------------------------" << endl;
      cout << m << endl;
      cout << "Est Carree                    : " << estCarree(m)    << endl;
      cout << "Est Reguliere                 : " << estReguliere(m) << endl;
      cout << "Longueur min. des vecteurs    : " << minCol(m)       << endl;
      cout << "Somme des lignes              : " << sommeLigne(m)   << endl;
      cout << "Somme des colonnes            : " << sommeColonne(m) << endl;
      cout << "Plus petite somme d'un vecteur : " << vectSommeMin(m) << endl;
      cout << "Melange de la matrice         : " << endl;
      shuffleMatrice(m);
      cout << m                                   << endl;
      cout << "Tri de la matrice             : " << endl;
      sortMatrice(m);
      cout << m                                   << endl;
      cout << "---------------------------------" << endl;
   }

   cout << "Presser ENTER pour quitter";
   VIDER_BUFFER;
   return EXIT_SUCCESS;
}



/*
-------------------------------------------------------------------------------
Nom du fichier : matrice.cpp
Nom du labo    : Labo 7 : Vecteur et Matrice
Auteur(s)      : Tim Ernst et Antonio Pollino
```

```
Date creation  : 08.12.2021
Description    : Fichier contenant les prototypes des fonctions du fichier
                 matrice.cpp.
Remarque(s)    : -
Compilateur    : Mingw-w64 g++ 11.2.0
-------------------------------------------------------------------------------
*/

#ifndef LABO7_MATRICES_MATRICE_H
#define LABO7_MATRICES_MATRICE_H
#include <vector>
#include <iostream>

using Vecteur = std::vector<int>;
using Matrice = std::vector<Vecteur>;

/// Nom         operator<<
/// But         Afficher un Vecteur au format (v1, v2, …, vn)
///
/// \param os
/// \param v    Vecteur à afficher
/// \return     Retourne le flux avec les données
std::ostream& operator<< (std::ostream& os, const Vecteur& v);

/// Nom         operator<<
/// But         Afficher une Matrice au format [(..),(..),(..)]
///
/// \param os
/// \param m    Matrice à afficher
/// \return     Retourne le flux avec les données
std::ostream& operator<< (std::ostream& os, const Matrice& m);

/// Nom         estCarre
/// But         Définir si une matrice est carrée
///
/// \param m    La matrice sur laquelle les tests seront effectués
/// \return     Retourne un booléen indiquant si la matrice est carrée
bool estCarree (const Matrice& m);

/// Nom         estReguliere
/// But         Définir si les lignes de la matrice ont la même taille
///
/// \param m    La matrice sur laquelle les tests seront effectués
/// \return     Retourne un booléen indiquant si la matrice est régulière
///             (toutes les lignes de même taille)
bool estReguliere (const Matrice& m);

/// Nom         minCol
/// But         Définir la longueur minimum des vecteurs d'une matrice
///
/// \param m    La matrice dont lla longueur des vecteurs sera contrôlée
/// \return     Retourne la longueur minimum des vecteurs d'une matrice
size_t minCol (const Matrice& m);

/// Nom         sommeLigne
/// But         Calculer la somme des valeurs d'une ligne comprise dans une matrice
///
/// \param m    La matrice dont les valeurs des lignes vont être additionnées
/// \return     Retourne un vecteur contenant la somme des valeurs de
///             chacune des lignes.
Vecteur sommeLigne (const Matrice& m);

/// Nom         sommeColonne
/// But         Calculer la somme des valeur d'une colonne comprise dans une matrice
///
/// \param m    La matrice dont les valeurs des colonnes vont être additionnées
/// \return     Retourne un vecteur contenant la somme des valeurs de
///             chacune des colonnes.
Vecteur sommeColonne (const Matrice& m);

/// Nom         vectSommeMin
/// But         Calculer la ligne d'une matrice dont la somme de ces éléments est
///             la plus faible
///
/// \param m    La matrice dont les lignes vont être additionnées
/// \return     Retour le vecteur d'une matrice dont la somme des valeurs
///             est la plus faible
```

```
Vecteur vectSommeMin (const Matrice& m);

/// Nom         shuffleMatrice
/// But         Mélanger les vecteurs d'une matrice sans altérer les vecteurs.
///
/// \param m    La matrice dont les vecteurs vont être mélangés
void shuffleMatrice (Matrice& m);

/// Nom         sortMatrice
/// But         Trier dans l'ordre croissant une matrice en fonction de
///             l'élément minimum d'un vecteur
///
/// \param m    La matrice dont les vecteurs vont être trié dans l'ordre croissant
void sortMatrice (Matrice& m);

#endif //LABO7_MATRICES_MATRICE_H



/*
-------------------------------------------------------------------------------
Nom du fichier : matrice.cpp
Nom du labo    : Labo 7 : Vecteur et Matrice
Auteur(s)      : Tim Ernst et Antonio Pollino
Date creation  : 08.12.2021
Description    : Fichier contenant les fonctions permettant de réaliser des
                 opérations sur les vecteurs et matrices.
Remarque(s)    : Ce fichier contient les fonctions suivantes :
                  - estCarre
                  - estReguliere
                  - minCol
                  - sommeLigne
                  - sommeColonne
                  - vectSommeMin
                  - shuffleMatrice
                  - sortMatrice
Compilateur    : Mingw-w64 g++ 11.2.0
-------------------------------------------------------------------------------
*/

#include <random>
#include <chrono>
#include <numeric>
#include <algorithm>
#include "matrice.h"

//-----------------------------------------------------------------
// Prototypes des fonctions non-utilisables par l'utilisateur
//-----------------------------------------------------------------

/// Nom         estDeTailleEgale
/// But         Comparer deux vecteurs afin de définir s'ils sont égaux
///
/// \param v1   Vecteur 1
/// \param v2   Vecteur 2
/// \return     Booléen permettant de définir s'ils sont égaux
bool estDeTailleEgale(const Vecteur& v1, const Vecteur& v2);

/// Nom         estLePlusPetitElement
/// But         Défini si l'élément le plus petit du vecteur 1 est plus petit
///             que l'élément du vecteur 2
///
/// \param v1   Vecteur 1
/// \param v2   Vecteur 2
/// \return     True si vecteur < vecteur 2
bool estLePlusPetitElement(const Vecteur& v1, const Vecteur& v2);

/// Nom         estPlusPetit
/// But         Permet de définir quel vecteur est le plus petit
///
/// \param v1   Vecteur 1
/// \param v2   Vecteur 2
/// \return     Booléen permettant de savoir si le vecteur1 est plus petit que le
///             vecteur 2
bool estPlusPetit(const Vecteur& v1, const Vecteur& v2);

/// Nom         sommeVecteur
```

```
/// But         Permet d'additionner tous les éléments d'un vecteur ou d'une ligne
///             de matrice
///
/// \param v    Vecteur contenant les éléments qui doivent être additionné
/// \return     La somme des éléments
int  sommeVecteur(const Vecteur& v);

/// Nom         sommeVecteurColonne
/// But         Permet d'additionner tous les éléments d'une colonne inclue dans
///             une matrice quelconque
///
/// \param m    Matrice dont les colonnes doivent être additionnée
/// \return     La somme des éléments
int sommeVecteurColonne(const Matrice& m, unsigned long long colonne);

/// Nom         maxCol
/// But         Définir le nombre de colonnes contenues dans une matrice
///
/// \param m    Matrice dont le nombre de colonnes doit être défini
/// \return     Le nombre de colonnes
size_t maxCol(const Matrice& m);

/// Nom         sommeVectPlusPetit
/// But         Définir quelle somme de vecteur est la plus faible
///
/// \param v1   Vecteur 1
/// \param v2   Vecteur 2
/// \return     Booléen permettant de définir si la somme des éléments du vecteur 1
///             est plus petite que celle du vecteur 2
bool sommeVectPlusPetit (const Vecteur& v1, const Vecteur& v2);

using namespace std;

//-----------------------------------------------------------------
// Fonctions non-utilisables par l'utilisateur
//-----------------------------------------------------------------
bool estDeTailleEgale(const Vecteur& v1, const Vecteur& v2){
   return v1.size() == v2.size();
}

bool estLePlusPetitElement(const Vecteur& v1, const Vecteur& v2){
   if(v1.empty()) return true;
   if(v2.empty()) return false;
   return *min_element(v1.begin(), v1.end()) < *min_element(v2.begin(), v2.end());
}

bool estPlusPetit(const Vecteur& v1, const Vecteur& v2){
   return v1.size() < v2.size();
}

int sommeVecteur(const Vecteur& v){
   return accumulate(v.begin(), v.end(), 0);
}

size_t maxCol(const Matrice& m){
   return max_element(m.begin(), m.end(), estPlusPetit)
      ->size();
}

int sommeVecteurColonne(const Matrice& m, unsigned long long colonne){
   int somme = 0;
   for (size_t i = 0; i < m.size(); ++i){
      if (!m[i].empty() && colonne < m[i].size()){
         somme += m[i][colonne];
      }
      else{
         continue;
      }
   }
   return somme;
}

bool sommeVectPlusPetit (const Vecteur& v1, const Vecteur& v2){
   return accumulate(v1.begin(), v1.end(),0) < accumulate(v2.begin(), v2.end(), 0);
}

//-----------------------------------------------------------------
```

```
// Fonctions utilisables par l'utilisateur
//-----------------------------------------------------------------

std::ostream& operator<< (ostream& os, const Vecteur& v){
   os << "(";
   for (Vecteur::const_iterator i = v.begin(); i < v.end(); ++i) {
      if (i != v.begin()){
         os << " ,";
      }
      os << *i;
   }
   os << ")";
   return os;
}

ostream& operator<< (ostream& os, const Matrice& m){
   os << "[";
   for(Matrice::const_iterator i = m.begin(); i < m.end(); ++i) {
      if (i != m.begin()){
         os << " ,";
      }
      os << *i;
   }
   os << "]";
   return os;
}

bool estCarree (const Matrice& m){
   if(m.empty()){
      return true;
   }
   return estReguliere(m) && m.size() == m[0].size();
}

bool estReguliere(const Matrice& m){
   if (m.empty()){
      return true;
   }
      return equal(m.begin(), m.end() - 1, m.begin() + 1,
                   estDeTailleEgale);
}

size_t minCol(const Matrice& m){
   if (!m.empty()){
      return min_element(m.begin(), m.end(), estPlusPetit)
         ->size();
   }
   return 0;
}

Vecteur sommeLigne(const Matrice& m){
   Vecteur v(m.size());
   if (!m.empty()){
      transform(m.begin(), m.end(), v.begin(), sommeVecteur);
   }
   return v;
}

Vecteur sommeColonne(const Matrice& m){
   Vecteur v(maxCol(m));

   for (unsigned long long i = 0; i < maxCol(m); ++i){
      v[i] = (sommeVecteurColonne(m,i));
   }
   return v;
}

Vecteur vectSommeMin (const Matrice& m){
   if (!m.empty()){
      return *min_element(m.begin(), m.end(), sommeVectPlusPetit);
   }
   return Vecteur{};
}

void shuffleMatrice (Matrice& m){
   unsigned seed = (unsigned)chrono::system_clock::now().time_since_epoch().count();
   shuffle(m.begin(),m.end(),default_random_engine(seed));
```

```
}

void sortMatrice (Matrice& m){
   sort(m.begin(),m.end(), estLePlusPetitElement);
}
```