```
/*
-----------------------------------------------------------------------------
Nom du fichier : main.cpp
Nom du labo    : Labo07_vecteur_matrice

Auteur(s)      : Baume Oscar & Guyot Grégoire
Date creation  : 08.12.2021

Description    : Quelques matrice pour tester les fonctions de la bibliothèque
                 matrice.

Remarque(s)    :

Modification:       ---
                    Date   :
                    Auteur :
                    Raison :

Compilateur    : Mingw-w64 g++ 11.2.0
-----------------------------------------------------------------------------
*/

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

using namespace std;

void testerMatrice(const Matrice& m);

int main() {
    // Quelques tests pour montrer le fonctionnement de matrice.h
    Matrice m;
    testerMatrice(m);

    m = {{4,4}, {1,3},{2}};
    testerMatrice(m);

    m = {{1,0,0}, {0,1,0},{0,0,1}};
    testerMatrice(m);

    m = {{1,2,3}, {1,2,3}, {1,2}, {1,2,3}};
    testerMatrice(m);

    m = {{1,2,3,4,5}, {1,2,3}, {12}, {1,2,0,0}, {1,2,3}};
    testerMatrice(m);

    m = {{9,8,7}, {6,5,4}, {3,2,1}};
    testerMatrice(m);

    return EXIT_SUCCESS;
}

void testerMatrice(const Matrice& m){
   cout   << "-------------------------" << boolalpha << endl
          << m << endl
          << "La matrice est carree                  : " << estCarree(m)      << endl
          << "La matrice est reguliere               : " << estReguliere(m)   << endl
          << "Taille min d'une ligne                 : " << minCol(m)         << endl
          << "Somme des lignes                       : " << sommeLigne(m)     << endl
          << "Somme des colonnes                     : " << sommeColonne(m)   << endl
          << "Vecteur des sommes minimal d'une ligne : " << vectSommeMin(m)   << endl
          << "Shuffle Matrice                        : " << shuffleMatrice(m) << endl
          << "Sort                                   : " << sortMatrice(m)    << endl;
}

/*
-----------------------------------------------------------------------------
Nom du fichier      : matrice.h
Nom du labo         : Labo07_vecteur_matrice

Auteur(s)           : Baume Oscar & Guyot Grégoire
Date creation       : 08.12.2021

Description         : Déclaration des fonctions de la bibliothèque matrice
Remarque(s)         :

Modification:       ---
```

```
                    Date   :
                    Auteur :
                    Raison :

Compilateur         : Mingw-w64 g++ 11.2.0
-------------------------------------------------------------------------------
*/

#ifndef LABO_07_VECTEUR_MATRICE_MATRICE_H
#define LABO_07_VECTEUR_MATRICE_MATRICE_H

#include <vector>
#include <iostream>

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

/**
 * Nom          : <<
 * Description  : opérateur de flux pour un Vecteur,
 *                affiche un Vecteur comme ceci (v1, v2, ... , vn)
 * Remarques    : n/a
 * @param os    : flux
 * @param v     : vecteur à afficher
 * @return      : retourne le flux
 */
std::ostream& operator << (std::ostream& os, const Vecteur& v);

/**
 * Nom          : <<
 * Description  : opérateur de flux pour une Matrice
 *                affiche une Matrice comme ceci [(Vecteur1), ... ,(Vecteurn)] *
 * Remarques    : n/a
 * @param os    : flux
 * @param m     : matrice à afficher
 * @return      : retourne le flux
 */
std::ostream& operator << (std::ostream& os, const Matrice& m);
/**
 * Nom          : estCaree
 * Description  : Permet de vérifier si la matrice en paramètre est carrée
 * Remarques    : Une matrice vide est considérée comme carrée
 * @param m     : La matrice à évaluer
 * @return      : Retourne un booléen true si la matrice est carrée
 */
bool estCarree(const Matrice& m);

/**
 * Nom          : estReguliere
 * Description  : Retourne un booléen indiquant si la matrice est régulière 2
 *                (toutes les lignes de même taille)
 * Remarques    : Une matrice vide est considérée comme régulière
 * @param m     : La matrice a évaluer
 * @return      : Retourne un booléen true si la matrice est régulière
 */
bool estReguliere(const Matrice& m);

/**
 * Nom          : minCol
 * Description  : Trouver la taille de la colonne la plus petite
 * Remarques    : n/a
 * @param m     : La matrice à évaluer
 * @return      : Retourne la longueur minimum des vecteurs d'une matrice
 */
size_t minCol(const Matrice& m);

/**
 * Nom          : sommeLigne
 * Description  : Retourne un vecteur contenant la somme des valeurs de
 *                chacune des lignes.
 * Remarques    : n/a
 * @param m     : La matrice a évaluer
 * @return      : Somme des lignes de la matrice en vecteur
 */
Vecteur sommeLigne(const Matrice& m);

/**
 * Nom          : sommeLigne
```

```
 * Description  : Retourne un vecteur contenant la somme des valeurs de
 *                chacune des lignes.
 * Remarques    :
 * @param m     : La matrice a évaluer
 * @return      : Somme des lignes de la matrice en vecteur
 */
Vecteur sommeColonne(const Matrice& m);

/**
 * Nom          : vectSommeMin
 * Description  : Retour le vecteur d'une matrice dont la somme des valeurs est la
 *                plus faible.
 * Remarques    : Si plusieurs vecteurs présentent la même somme, la fonction
 *                retourne celui d'indice le plus faible
 * @param m     : La matrice a évaluer
 * @return      : Le vecteur de la matrice ayant la somme minimum
 */
Vecteur vectSommeMin(const Matrice& m);

/**
 * Nom          : shuffleMatrice
 * Description  : Mélange les vecteurs d'une matrice
 * Remarques    : Les valeurs se trouvant dans les vecteurs ne sont pas altérées
 * @param m     : La matrice à évaluer
 * @return      : Une matrice dont les vecteurs ont été déplacés de manière aléatoire
 */
Matrice shuffleMatrice(const Matrice& m);

/**
 * Nom          : sortMatrice
 * Description  : Trier dans l'ordre croissant une matrice en fonction de
 *                l'élément min d'un vecteur.
 * Remarques    :
 * @param m     : La matrice à trier
 * @return      : La matrice triée
 */
Matrice sortMatrice(const Matrice& m);

#endif //LABO_07_VECTEUR_MATRICE_MATRICE_H
/*
-----------------------------------------------------------------------------
Nom du fichier      : matrice.cpp
Nom du labo         : Labo07_vecteur_matrice

Auteur(s)           : Baume Oscar & Guyot Grégoire
Date creation       : 08.12.2021

Description         : Déclaration des fonctions de la bibliothèque matrice
Remarque(s)         :

Modification:       ---
                    Date   :
                    Auteur :
                    Raison :

Compilateur         : Mingw-w64 g++ 11.2.0
-----------------------------------------------------------------------------
*/
#include "matrice.h"
#include <algorithm>
#include <numeric>
#include <chrono>
#include <random>

using namespace std;

/**
 * Nom          : compSortMatrice
 * Description  : Compare le plus petit élément de v1 avec le plus petit élément
 *                de v2
 * Remarques    : Utiliser pour la fonction sortMatrice
 * @param v1    : premier vecteur
 * @param v2    : deuxième vecteur
 * @return      : si min_element(v1) < min_element(v2)
 */
bool compSortMatrice(const Vecteur& v1, const Vecteur& v2);

/**
```

```
 * Nom          : vecteurPlusPetit
 * Description  : Compare si v1 et plus petit que v2
 * Remarques    : Utiliser pour les fonctions estReguliere et estCarree
 * @param v1    : premier vecteur
 * @param v2    : deuxiême vecteur
 * @return      : si v1.size() < v2.size()
 */
bool vecteurPlusPetit(const Vecteur &v1, const Vecteur &v2);

bool estCarree(const Matrice& m) {
    if(m.empty())
        return true;
   return (*min_element(m.begin(), m.end(), vecteurPlusPetit)).size() == m.size() &&
          (*max_element(m.begin(), m.end(), vecteurPlusPetit)).size() == m.size();
}

bool estReguliere(const Matrice& m) {
   if(m.empty()) {
        // retourne vrai si la matrice est vide
        return true;
    }
   // retourne si le plus petit vecteur à la même taille que le plus grand
   return   (*min_element(m.begin(), m.end(), vecteurPlusPetit)).size() ==
            (*max_element(m.begin(), m.end(), vecteurPlusPetit)).size();
}

// pointe sur vecteur le plus petit et retourne ca taille
size_t minCol(const Matrice& m) {
    if (m.empty()) {
        return 0;
    }
   return (*min_element(m.begin(), m.end(), vecteurPlusPetit)).size();
}

Vecteur sommeLigne(const Matrice& m) {
   if(m.empty())
      return {};
   Vecteur vOut;

   for(Matrice::const_iterator i = m.cbegin(); i != m.cend(); ++i) {
      // pour chaque vecteur de la matrice
      // on utilise accumulate pour additionner tout les éléments du vecteur
      // et on push_back le résultat de accumulate dans le vecteur
      vOut.push_back(accumulate(i->cbegin(),i->cend(),0));
   }
   return vOut;
}

Vecteur sommeColonne(const Matrice& m) {
   if(m.empty())
      return {};
   Vecteur sommeColonne(m.size());

   for (const Vecteur &i : m) {
      for (size_t j = 0; j < i.size(); j++) {
         sommeColonne[j] += i[j];
      }
   }
   return sommeColonne;
}

Vecteur vectSommeMin(const Matrice& m) {
   if(m.empty()){
      return {};
   }
   // on prend le vecteur de la somme des ligne de la matrice
   Vecteur sommeLigne = ::sommeLigne(m);
   // on retourne le vecteur à la position entre le début de sommeLigne et
   // l'élément minimum de sommeLigne
   return m.at(distance(sommeLigne.begin(),
                                min_element(sommeLigne.begin(), sommeLigne.end())));
}

Matrice shuffleMatrice(const Matrice& m) { // verifier par copie ou reference
   if(m.empty())
      return {};
   unsigned seed = (unsigned)chrono::system_clock::now().time_since_epoch().count();
   Matrice mOut = m;
```

```
   shuffle(mOut.begin(), mOut.end(), default_random_engine(seed));

   return mOut;
}

Matrice sortMatrice(const Matrice& m) {
   if(m.empty())
      return {};
   Matrice mOut = m;
   // on tri mOut avec compSortMatrice
   sort(mOut.begin(),mOut.end(), compSortMatrice);
   return mOut;
}

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

bool compSortMatrice(const Vecteur& v1, const Vecteur& v2) {
   // la valeur minimal de v1
   int min_v1 = *min_element(v1.begin(), v1.end());
   // la valeur minimal de v2
   int min_v2 = *min_element(v2.begin(), v2.end());
   if(min_v1 < min_v2)
      return true;
   else if(min_v1 > min_v2)
      return false;
   // sinon si la taille de v1 est plus petite ou égale à la taille de v2
   else if(v1.size() <= v2.size())
      return true;
   else
      return false;
}

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

ostream& operator<< (ostream& os, const Matrice& m) {
   cout << "[";
   for(Matrice::const_iterator i = m.cbegin();i != m.cend(); ++i) {
      if(i != m.begin())
         cout << ", ";
      cout << "(";
      for(Vecteur::const_iterator j = i->cbegin(); j != i->cend(); ++j) {
         if(j != i->begin())
            cout << ", ";
         cout << *j;
      }
      cout << ")";
   }
   cout << "]";
   return os;
}
```