```
/*
-------------------------------------------------------------------------------
File name     : main.cpp
Lab name      : Lab 7, Vectors and matrices
Authors       : Miguel Jalube, Leandro Saraiva Maia
Creation date : 07.12.2021

Description   : This program tests the functions given by the matrixUtilites
                library.
Remark(s)     : An empty matrix is considered as square and regular. irregular
                matrices are ignored.

Compiler      : Mingw-w64 g++ 11.1.0
-------------------------------------------------------------------------------
*/

#include <cstdlib>            // required for EXIT_SUCCESS
#include <iostream>           // required for cout
#include <limits>             // required for numeric_limits<...>
#include <vector>             // required because of hidden use of vector
#include "matrixUtilities.h"  // required for matrix manipulations

using namespace std ;

void test(Matrix& matrix);

int main() {
   cout << boolalpha;

   Matrix testMatrices[] = {
      // ----- Square cases -----
      {{ 1, 4,-2},
       { 2, 2, 2},
       {-1,-1, 3}},

      {{ 3,-1, 2, 1},
       { 4, 1, 0,-2},
       { 0, 1, 3,-4},
       {-3, 0, 2,-4}},

      {{1,5},
       {3,1}},

      {{ 0, 0, 0},
       { 0, 0, 0},
       { 0, 0, 0}},

      // ----- Regular cases -----
      {{ 1, 2, 2},
       { 0, 1,-1},
       { 0, 0, 2},
       { 4, 5, 6}},

      {{ 1, 3, 2, 4},
       { 0, 1,-1, 4}},

      {{ 1, 2, 3, 4, 5}},

      {{ 0, 0, 0, 0},
       { 0, 0, 0, 0}},

      // ----- Irregular cases -----
      {{ 1, 4},
       { 0, 3, 0},
       { 1, 5, 3}},

      {{ 3, 0},
       { 1,-1},
       {-5}},

      {{ 4, 0, -1, 1, -1, 0},
       { 1,-1},
```

```
 73          { 2, 1}},
 74
 75         // ----- Empty cases -----
 76         {},
 77
 78         {{}},
 79
 80         {{},
 81          {}},
 82
 83         {{0,0,0},
 84          {},
 85          {0,0}},
 86
 87         {},
 88         {{}},
 89         {{0}},
 90      };
 91
 92      for (Matrix& testMatrix : testMatrices) {
 93         test(testMatrix);
 94      }
 95
 96      //----- End of program -----
 97      cout << "Press ENTER to quit.";
 98      cin.ignore(numeric_limits<streamsize>::max(), '\n'); // empty buffer
 99      return EXIT_SUCCESS;
100   }
101
102   void test(Matrix& matrix){
103      cout << "Display vector       : ";
104      if(matrix.empty()) {
105         cout << "()";
106      }else{
107         cout << matrix.at(0);
108      }
109      cout << endl;
110      cout << "Display matrix       : " << matrix << endl;
111      cout << "Is square            : " << isSquare(matrix) << endl;
112      cout << "Is regular           : " << isRegular(matrix) << endl;
113      cout << "Minimum row size     : " << minRow(matrix) << endl;
114      cout << "Row sum              : " << sumRow(matrix) << endl;
115      cout << "Column sum           : " << sumColumn(matrix) << endl;
116      cout << "Vector minimal sum   : " << vectSumMin(matrix) << endl;
117      shuffleMatrix(matrix);
118      cout << "Matrix after shuffle : " << matrix << endl;
119      sortMatrix(matrix);
120      cout << "Matrix after sort    : " << matrix << endl;
121      cout << "-------------------------------------------------------"<<endl;
122   }
```

```cpp
/*
-----------------------------------------------------------------------------
File name     : matrixUtilities.cpp
Authors       : Miguel Jalube, Leandro Saraiva Maia
Creation date : 07.12.2021

Description   : See matrixUtilities.h
Remark(s)     :

Compiler      : Mingw-w64 g++ 11.1.0
-----------------------------------------------------------------------------
*/

#include "matrixUtilities.h" // Prototypes and aliases (Matrix, VInt)
#include <vector>            // required for internal use of vector (Matrix, VInt)
#include <algorithm>         // required for sort functions
#include <ctime>             // required for random seed
#include <numeric>           // required for accumulate
#include <iterator>          // required for distance()
#include <iostream>          // required for cout
#include <ctime>             // required to generate a seed based on time
#include <random>            // required to generate random numbers

using namespace std;

// ----- predicates -----
/**
 * Checks if 2 vectors have the same size
 * @param vector1
 * @param vector2
 * @return boolean true if vectors have the same size
 */
bool isSameSizeAs(const VInt& vector1, const VInt& vector2) {
   return vector1.size() == vector2.size();
}

/**
 * Checks if vector1 smaller than vector2
 * @param vector1
 * @param vector2
 * @return boolean true if vector1 is smaller
 */
bool isSmallerThan(const VInt& vector1, const VInt& vector2){
   return vector1.size() < vector2.size();
}

/**
 * Checks if the first element of the vector1 is smaller than the first element of
 * the vector2. When one of the two vectors is empty, we sort it first
 * @param vector1
 * @param vector2
 * @return boolean true if the first value of the vector1 is smaller
 */
bool isFirstElemSmallerThan(const VInt& vector1, const VInt& vector2) {
   return (vector1.empty() || vector2.empty()) || vector1.front() < vector2.front();
}

// ----- utilities functions -----
bool isSquare(const Matrix& matrix){
   // Returns true if the matrix is empty, otherwise checks if the matrix is
   // regular AND if the size of a vector (since they are regular they have all
   // the same size, so we can take any vector) is equal to the size of the matrix
   return matrix.empty() ||
          (isRegular(matrix) && matrix.front().size() == matrix.size());
}

bool isRegular(const Matrix& matrix){
   // Returns true if the matrix is empty, otherwise checks if every element has
   // the same size as the next one. If this is true, then it means that every
   // vector has the same size
   return matrix.empty() || equal(matrix.begin(), matrix.end() - 1,
                                  matrix.begin() + 1, isSameSizeAs);
```

```
73  }
74
75  // ! This function is named "minCol" in the exercice, but we renamed it "minRow".
76  // We did it because it did not reflect what the function actually does
77  size_t minRow(const Matrix& matrix){
78     // Return 0 if matrix is empty, otherwise compare the size of every row and
79     // returns the size of the smallest
80     return matrix.empty() ? 0 :
81            (*min_element(matrix.begin(), matrix.end(), isSmallerThan)).size();
82  }
83
84  VInt sumRow(const Matrix& matrix){
85     if (matrix.empty()) return {};
86
87     VInt result = VInt(matrix.size());
88     // For each vector in the matrix, we sum the elements of the vector and put it
89     // in a vector that is indexed in the same way as matrix (that is why we use a
90     // "for int i" and not a "for auto i :", because we want the int index).
91     for (size_t i = 0; i < matrix.size(); ++i) {
92        result[i] = accumulate(matrix[i].begin(), matrix[i].end(), 0);
93     }
94
95     return result;
96  }
97
98  VInt sumColumn(const Matrix& matrix){
99     VInt result;
100    if(!matrix.empty()) {
101       // Resizes result vector to the same size as the biggest vector in the matrix
102       result.resize(
103          (*max_element(matrix.begin(), matrix.end(), isSmallerThan)).size());
104
105       // For each vector in the matrix increment the result vector (default value
106       // is 0) adding the value of matrix[i][j] to result[j]
107       for (const VInt &i: matrix) {
108          for (size_t j = 0; j < i.size(); ++j) {
109             result.at(j) += i.at(j);
110          }
111       }
112    }
113    return result;
114 }
115
116 VInt vectSumMin(const Matrix& matrix){
117    if (matrix.empty()) return {};
118
119    // First, get a vector containing the sum of each row (indexed identically as
120    // the matrix).
121    VInt vSumRow = sumRow(matrix);
122    // Then, we get index of the row with the smallest sum by getting the distance
123    // between the being iterator and the min_element iterator.
124    size_t indexOfMinSumRow = (size_t)distance(vSumRow.begin(),
125                              min_element(vSumRow.begin(), vSumRow.end()));
126    // Since vSumRow is indexed identically as the matrix, we can access the matrix
127    // with the same index
128    return matrix[indexOfMinSumRow];
129 }
130
131 void shuffleMatrix(Matrix& matrix){
132    if(!matrix.empty()) {
133       // Sets the random seed in static, so it is not reset for each call
134       static long long int seed = time(nullptr);
135
136       // Shuffles the matrix elements using the random generator mt19937 seeded using
137       // the ctime library
138       shuffle(matrix.begin(), matrix.end(), mt19937(seed));
139    }
140 }
141
142 void sortMatrix(Matrix& matrix){
143    if (matrix.empty()) return;
144
```

```
145      // Sort matrix by looking at the smallest first element of each row.
146      sort(matrix.begin(), matrix.end(), isFirstElemSmallerThan);
147   }
148
149   ostream& operator<< (ostream& os,const VInt& vector){
150      os << "(";
151      // Loop to send in the ostream, the value at *i
152      for(VInt::const_iterator i = vector.begin(); i != vector.end(); ++i){
153         if(i != vector.begin()){
154            os << ", ";
155         }
156         os << *i;
157      }
158      os << ")";
159      return os;
160   }
161
162   ostream& operator<< (ostream& os,const Matrix& matrix){
163      os << "[";
164      // Loop to send in the ostream, the vector at *i
165      for(Matrix::const_iterator i = matrix.begin(); i != matrix.end(); ++i){
166         if(i != matrix.begin()){
167            os << ", ";
168         }
169         os << *i;
170      }
171      os << "]";
172      return os;
173   }
```

```cpp
/*
-----------------------------------------------------------------------------
File name     : matrixUtilities.cpp
Authors       : Miguel Jalube, Leandro Saraiva Maia
Creation date : 07.12.2021

Description   : See matrixUtilities.h
Remark(s)     :

Compiler      : Mingw-w64 g++ 11.1.0
-----------------------------------------------------------------------------
*/

#include "matrixUtilities.h" // Prototypes and aliases (Matrix, VInt)
#include <vector>            // required for internal use of vector (Matrix, VInt)
#include <algorithm>         // required for sort functions
#include <ctime>             // required for random seed
#include <numeric>           // required for accumulate
#include <iterator>          // required for distance()
#include <iostream>          // required for cout
#include <ctime>             // required to generate a seed based on time
#include <random>            // required to generate random numbers

using namespace std;

// ----- predicates -----
/**
 * Checks if 2 vectors have the same size
 * @param vector1
 * @param vector2
 * @return boolean true if vectors have the same size
 */
bool isSameSizeAs(const VInt& vector1, const VInt& vector2) {
   return vector1.size() == vector2.size();
}

/**
 * Checks if vector1 smaller than vector2
 * @param vector1
 * @param vector2
 * @return boolean true if vector1 is smaller
 */
bool isSmallerThan(const VInt& vector1, const VInt& vector2){
   return vector1.size() < vector2.size();
}

/**
 * Checks if the first element of the vector1 is smaller than the first element of
 * the vector2. When one of the two vectors is empty, we sort it first
 * @param vector1
 * @param vector2
 * @return boolean true if the first value of the vector1 is smaller
 */
bool isFirstElemSmallerThan(const VInt& vector1, const VInt& vector2) {
   return (vector1.empty() || vector2.empty()) || vector1.front() < vector2.front();
}

// ----- utilities functions -----
bool isSquare(const Matrix& matrix){
   // Returns true if the matrix is empty, otherwise checks if the matrix is
   // regular AND if the size of a vector (since they are regular they have all
   // the same size, so we can take any vector) is equal to the size of the matrix
   return matrix.empty() ||
          (isRegular(matrix) && matrix.front().size() == matrix.size());
}

bool isRegular(const Matrix& matrix){
   // Returns true if the matrix is empty, otherwise checks if every element has
   // the same size as the next one. If this is true, then it means that every
   // vector has the same size
   return matrix.empty() || equal(matrix.begin(), matrix.end() - 1,
                                  matrix.begin() + 1, isSameSizeAs);
```

```
73  }
74
75  // ! This function is named "minCol" in the exercice, but we renamed it "minRow".
76  // We did it because it did not reflect what the function actually does
77  size_t minRow(const Matrix& matrix){
78     // Return 0 if matrix is empty, otherwise compare the size of every row and
79     // returns the size of the smallest
80     return matrix.empty() ? 0 :
81            (*min_element(matrix.begin(), matrix.end(), isSmallerThan)).size();
82  }
83
84  VInt sumRow(const Matrix& matrix){
85     if (matrix.empty()) return {};
86
87     VInt result = VInt(matrix.size());
88     // For each vector in the matrix, we sum the elements of the vector and put it
89     // in a vector that is indexed in the same way as matrix (that is why we use a
90     // "for int i" and not a "for auto i :", because we want the int index).
91     for (size_t i = 0; i < matrix.size(); ++i) {
92        result[i] = accumulate(matrix[i].begin(), matrix[i].end(), 0);
93     }
94
95     return result;
96  }
97
98  VInt sumColumn(const Matrix& matrix){
99     VInt result;
100    if(!matrix.empty()) {
101       // Resizes result vector to the same size as the biggest vector in the matrix
102       result.resize(
103          (*max_element(matrix.begin(), matrix.end(), isSmallerThan)).size());
104
105       // For each vector in the matrix increment the result vector (default value
106       // is 0) adding the value of matrix[i][j] to result[j]
107       for (const VInt &i: matrix) {
108          for (size_t j = 0; j < i.size(); ++j) {
109             result.at(j) += i.at(j);
110          }
111       }
112    }
113    return result;
114 }
115
116 VInt vectSumMin(const Matrix& matrix){
117    if (matrix.empty()) return {};
118
119    // First, get a vector containing the sum of each row (indexed identically as
120    // the matrix).
121    VInt vSumRow = sumRow(matrix);
122    // Then, we get index of the row with the smallest sum by getting the distance
123    // between the being iterator and the min_element iterator.
124    size_t indexOfMinSumRow = (size_t)distance(vSumRow.begin(),
125                              min_element(vSumRow.begin(), vSumRow.end()));
126    // Since vSumRow is indexed identically as the matrix, we can access the matrix
127    // with the same index
128    return matrix[indexOfMinSumRow];
129 }
130
131 void shuffleMatrix(Matrix& matrix){
132    if(!matrix.empty()) {
133       // Sets the random seed in static, so it is not reset for each call
134       static long long int seed = time(nullptr);
135
136       // Shuffles the matrix elements using the random generator mt19937 seeded using
137       // the ctime library
138       shuffle(matrix.begin(), matrix.end(), mt19937(seed));
139    }
140 }
141
142 void sortMatrix(Matrix& matrix){
143    if (matrix.empty()) return;
144
```

```
145      // Sort matrix by looking at the smallest first element of each row.
146      sort(matrix.begin(), matrix.end(), isFirstElemSmallerThan);
147   }
148
149   ostream& operator<< (ostream& os,const VInt& vector){
150      os << "(";
151      // Loop to send in the ostream, the value at *i
152      for(VInt::const_iterator i = vector.begin(); i != vector.end(); ++i){
153         if(i != vector.begin()){
154            os << ", ";
155         }
156         os << *i;
157      }
158      os << ")";
159      return os;
160   }
161
162   ostream& operator<< (ostream& os,const Matrix& matrix){
163      os << "[";
164      // Loop to send in the ostream, the vector at *i
165      for(Matrix::const_iterator i = matrix.begin(); i != matrix.end(); ++i){
166         if(i != matrix.begin()){
167            os << ", ";
168         }
169         os << *i;
170      }
171      os << "]";
172      return os;
173   }
```