```cpp
/*-----------------------------------------------------------------------------------
Filename       : main.cpp
Authors        : Maëlle Vogel and Tobie Praz
Creation date  : 01.12.2020
Description    : The program tests each functions created in library.cpp
                  - display a vector
                  - display a matrix
                  - check if a matrix is a square
                  - check if a matrix is regular
                  - return the size of the longest vector
                  - return a vector containing the sum of each vector
                  - return the vector with the smallest sum
                  - shuffle the vector order in the matrix
                  - sort the matrix by the biggest number in a vector
                  - sum the right to left diagonal /
                  - sum the left to right diagonal \
Compiler       : Mingw-w64 g++ 8.1.0
-----------------------------------------------------------------------------------*/

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

using namespace std;

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

int main() {

    //INIT MATRIX
    IntMatrix matrix1 = {{12, 2, 43},
                         {32, 2, 21}};
    IntMatrix matrix2 = {{1, 25, 2},
                         {3, 38, 1},
                         {4, 23, 1}};
    IntMatrix matrix3 = {{16, 2,  2},
                         {3,  34, 17, 5},
                         {4,  43, 1}};

    cout << boolalpha;
    cout << "-------------------------SQUARE-----------------------------------------" << endl;
    cout << matrix1 << " is square: " << isSquare(matrix1) << endl;
    cout << matrix2 << " is square: " << isSquare(matrix2) << endl;

    cout << "-------------------------REGULAR----------------------------------------" << endl;
    cout << matrix1 << " is regular: " << isRegular(matrix1) << endl;
    cout << matrix2 << " is regular: " << isRegular(matrix2) << endl;
    cout << matrix3 << " is regular: " << isRegular(matrix3) << endl;

    cout << "-------------------------LONGEST VECTOR---------------------------------" << endl;
    cout << matrix2 << " max vector size: " << maxCol(matrix2) << endl;
    cout << matrix3 << " max vector size: " << maxCol(matrix3) << endl;

    cout << "-------------------------SUM-OF-EACH-VECTOR-----------------------------" << endl;
    cout << matrix2 << " line sum: " << lineSum(matrix2) << endl;
    cout << matrix3 << " line sum: " << lineSum(matrix3) << endl;

    cout << "-------------------------SMALLEST-VECTOR-SUM----------------------------" << endl;
    cout << matrix2 << " vector with min line sum: " << vectMinSum(matrix2) << endl;
    cout << matrix3 << " vector with min line sum: " << vectMinSum(matrix3) << endl;

    cout << "-------------------------SHUFFLE-A-MATRIX-------------------------------" << endl;
    cout << matrix1 << " before shuffle" << endl;
    shuffleMatrix(matrix1);
    cout << matrix1 << " after shuffle"  << endl;
    cout << matrix2 << " before shuffle" << endl;
    shuffleMatrix(matrix2);
    cout << matrix2 << " after shuffle"  << endl;

    cout << "-------------------------SORT-BY-BIGGEST-NUMBER-IN-VECTOR---------------" << endl;
    cout << matrix1 << " before sort" << endl;
    sortMatrix(matrix1);
    cout << matrix1 << " after sort"  << endl;

    cout << "------------------------DIAGONAL----------------------------------------" << endl;
    int resultLR1;
    cout << matrix2 << " left to right diagonal exists: " << diagLRSum(matrix2, resultLR1)
         << ", result: " << resultLR1 << endl;

    int resultRL1;
    cout << matrix2 << " right to left diagonal exists: " << diagRLSum(matrix2, resultRL1)
         << ", result: " << resultRL1 << endl;

    int resultLR2;
    cout << matrix1 << " left to right diagonal exists: " << diagLRSum(matrix1, resultLR2)
         << ", result: " << resultLR2 << endl;
    cout << endl;

    cout << "ENTER FOR EXIT";
    EMPTY_BUFFER
    return EXIT_SUCCESS;
}
```

```cpp
#ifndef LABO5_VECTEURS_LIBRARY_H
#define LABO5_VECTEURS_LIBRARY_H

#include <string>
#include <vector>

using namespace std;

using IntVector = vector<int>;
using IntMatrix = vector<IntVector>;

ostream &operator<<(ostream &os, const IntVector &v);
ostream &operator<<(ostream &os, const IntMatrix &m);

/**
 * Is the matrix a square ? (N x N)
 * @param matrix: the matrix to analyse
 * @return True if the matrix is a square,
 *         false if not
 */
bool isSquare(const IntMatrix &matrix);

/**
 * Is the matrix regular ? (All lines same size)
 * @param matrix: the matrix to check
 * @return True if the matrix is regular,
 *         false if not
 */
bool isRegular(const IntMatrix &matrix);

/**
 * Returns the size of the longest vector of a matrix
 * @param matrix: the matrix to analyse
 * @return Longest line size
 */
int maxCol(const IntMatrix &matrix);

/**
 * Returns a vector containing the sum of the values of each lines.
 * @param matrix: the matrix containing the vectors to sum
 * @return Vector of all line sums
 */
IntVector lineSum(const IntMatrix &matrix);

/**
 * Returns the vector of a matrix with the lowest sum of values.
 * If several vectors have the same sum, the function returns the one with the lowest index
 * @param matrix: the matrix to analyse
 * @return Line with smallest sum
 */
IntVector vectMinSum(const IntMatrix &matrix);

/**
 * Shuffles the vectors of a matrix without changing the vectors
 * @param matrix: the matrix to shuffle
 */
void shuffleMatrix(IntMatrix &matrix);

/**
 * Sorts a matrix (reverse order of the biggest line element)
 * @param matrix: the matrix to sort
 */
void sortMatrix(IntMatrix &matrix);

/**
 * Computes the left to right diagonal sum and returns true if the matrix is valid (is square)
 * @param matrix: the matrix to sum the diagonal
 * @param result: where left to right diagonal sum will be stored
 * @return True if the diagonal exists,
 *         false if not
 */
bool diagLRSum(const IntMatrix &matrix, int &result);
```

```

/**
 * Computes the right to left diagonal sum and returns true if the matrix is valid (is square)
 * @param matrix: the matrix to sum the diagonal
 * @param result: where right to left diagonal sum will be stored
 * @return True if the diagonal exists,
 *         false if not
 */
bool diagRLSum(const IntMatrix &matrix, int &result);

#endif //LABO5_VECTEURS_LIBRARY_H

```

```cpp
/*-----------------------------------------------------------------------------
Filename       : main.cpp
Authors        : Maëlle Vogel and Tobie Praz
Creation date  : 01.12.2020
Description    : This library provides functions to:
                   - display a vector
                   - display a matrix
                   - check if a matrix is a square
                   - check if a matrix is regular
                   - return the size of the longest vector
                   - return a vector containing the sum of each vector
                   - return the vector with the smallest sum
                   - shuffle the vector order in the matrix
                   - sort the matrix by the biggest number in a vector
                   - sum the right to left diagonal /
                   - sum the left to right diagonal \
Compiler       : Mingw-w64 g++ 8.1.0
-----------------------------------------------------------------------------*/
#include "library.h"

#include <iostream>
#include <string>
#include <cctype>
#include <algorithm>
#include <numeric>
#include <chrono>
#include <random>

using namespace std;

//Utility functions
string commaJoinInt(const string &a, int b) {
    return a + ", " + to_string(b);
}

string commaJoinString(const string &a, const string &b) {
    return a + ", " + b;
}

string vecToString(const IntVector &v) {
    string vec;
    if(!v.empty()) {
        //Join vector elements with ", "
        vec = accumulate(next(v.begin()), v.end(), to_string(v[0]), commaJoinInt);
    }
    return "(" + vec + ")";
}

int vecSize(const IntVector &v) {
    return v.size();
}

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

bool comparator(const IntVector &a, const IntVector &b) {
    return max_element(a.begin(), a.end()) > max_element(b.begin(), b.end());
}

//Implementations
ostream &operator<<(ostream &os, const IntVector &v) {
    cout << vecToString(v);
    return os;
}

ostream &operator<<(ostream &os, const IntMatrix &m) {
    vector<string> strings(m.size());
    //Convert matrix lines to string
    transform(m.begin(), m.end(), strings.begin(), vecToString);

    string mat;
    if(!strings.empty()) {
        //Join matrix lines with ", "
        mat = accumulate(next(strings.begin()), strings.end(), strings[0], commaJoinString);
    }

    cout << "[" << mat << "]";
    return os;
}

bool isRegular(const IntMatrix &matrix) {
    int size = matrix.size();
    if(size) {
        IntVector sizes(size);
        //Get lines sizes
        transform(matrix.begin(), matrix.end(), sizes.begin(), vecSize);
        //Count the number of lines with same size as first line
        return count(sizes.begin(), sizes.end(), sizes[0]) == size;
    }
    return true;
}

bool isSquare(const IntMatrix &matrix) {
    return matrix.empty() || isRegular(matrix) && matrix[0].size() == matrix.size();
}

int maxCol(const IntMatrix &matrix) {
    IntVector sizes(matrix.size());
    //Get lines sizes
    transform(matrix.begin(), matrix.end(), sizes.begin(), vecSize);
    //Fin biggest size
    return *max_element(sizes.begin(), sizes.end());
}

IntVector lineSum(const IntMatrix &matrix) {
    IntVector result(matrix.size());
    transform(matrix.begin(), matrix.end(), result.begin(), sum);
    return result;
}

IntVector vectMinSum(const IntMatrix &matrix) {
    IntVector sum = lineSum(matrix);
    //Get min line sum iterator and compute index
    int i = min_element(sum.begin(), sum.end()) - sum.begin();
    return matrix[i];
}

void shuffleMatrix(IntMatrix &matrix) {
    unsigned seed = chrono::system_clock::now().time_since_epoch().count();
    shuffle(matrix.begin(), matrix.end(), default_random_engine(seed));
}

void sortMatrix(IntMatrix &matrix) {
    sort(matrix.begin(), matrix.end(), comparator);
}

bool diagRLSum(const IntMatrix &matrix, int &result) {
    result = 0;
    if (isSquare(matrix)) {
        for (size_t i = 0; i < matrix.size(); ++i) {
            result += matrix[i][matrix.size() - i - 1];
        }
        return true;
    }
    return false;
}

bool diagLRSum(const IntMatrix &matrix, int &result) {
    result = 0;
    if (isSquare(matrix)) {
        for (size_t i = 0; i < matrix.size(); ++i) {
            result += matrix[i][i];
        }
```

```
            return true;
        }
        return false;
    }



```