Contents

1 Basic Test Results

```
List of submitted files:
Makefile
Map.h
Matrix.cpp
Matrix.h
MyMatrix.cpp
MyMatrix.h
RegMatrix.cpp
RegMatrix.h
SparseMatrix.cpp
SparseMatrix.h
Vector.h

Checking MyMatrix.cpp
---------------------
MyMatrix.cpp file exists

Checking MyMatrix.h
----------------------
MyMatrix.h file exists
cp: omitting directory /cs/course/2014/slabcpp/public/ex2/code_files/strategy_example

Checking Makefile
----------------------
Makefile exists
make...
g++ -Wall -std=c++11 -c MyMatrix.cpp
g++ -Wall -std=c++11 -c Matrix.cpp
g++ -Wall -std=c++11 -c RegMatrix.cpp
g++ -Wall -std=c++11 -c SparseMatrix.cpp
g++ -Wall -std=c++11 MyMatrix.cpp Matrix.cpp RegMatrix.cpp SparseMatrix.cpp DemoMyMatrix.cpp -o DemoMyMatrix

Testing DemoMyMatrix
----------------------
OK
Done running the test.

NOTE: The presubmission script does not stop on error. You need to go over its output
      and check that there are no ERRORs lines.

========================
= Checking coding style =
========================
 ** Total Violated Rules      : 0
 ** Total Errors Occurs       : 0
 ** Total Violated Files Count: 0
```

2 Makefile

```
CC = g++ -Wall -std=c++11

defaulf: all demo

all: MyMatrix.o Matrix.o RegMatrix.o SparseMatrix.o

demo:
    $(CC) MyMatrix.cpp Matrix.cpp RegMatrix.cpp SparseMatrix.cpp DemoMyMatrix.cpp -o DemoMyMatrix

MyMatrix.o: MyMatrix.cpp MyMatrix.h Matrix.h Map.h Vector.h
    $(CC) -c MyMatrix.cpp

Matrix.o: Matrix.cpp Matrix.h
    $(CC) -c Matrix.cpp

RegMatrix.o: RegMatrix.cpp RegMatrix.h Matrix.h Vector.h
    $(CC) -c RegMatrix.cpp

SparseMatrix.o: SparseMatrix.cpp SparseMatrix.h Matrix.h Map.h
    $(CC) -c SparseMatrix.cpp

matrix:
    $(CC) MyMatrix.cpp Matrix.cpp RegMatrix.cpp SparseMatrix.cpp main.cpp -o matrix
    matrix
    matrix > matrix.out
    FC matrix.out roitest.out

test:
    $(CC) MyMatrix.cpp Matrix.cpp RegMatrix.cpp SparseMatrix.cpp test.cpp -o test
    test > test.out
    FC test.out test_result

tar:
    tar cvf ex2.tar MyMatrix.cpp Matrix.cpp RegMatrix.cpp SparseMatrix.cpp \
            MyMatrix.h Matrix.h RegMatrix.h SparseMatrix.h \
            Map.h Vector.h Makefile

clean:
    rm -f *.o demo

.PHONY: all tar clean test default matrix
```

3 Map.h

```
/**
 * A map from unsigend int to double.
 * This is simply a wrapper, or a typedef of the map STL
 * Support iterators, assingment, and more methods.
 */
#ifndef _MAP_H_
#define _MAP_H_

#include <map>
#include <utility>

typedef std::pair<unsigned int, unsigned int> Pair;
typedef std::map<Pair, double> Map;

#endif // _MAP_H_
```

4 Matrix.h

```
/**
* Matrix.h
*
* ------------------------------------------------------------------------------------------------
* General:        This class is an abstract class represents the acuall matrix variable
*                 The matrix can be in 2 different way. regular and sparse.
*                 regular - more or equal to half of the values are nor zero, contained in a vector
*                 sparse - less than half are zero, contained in a map.
*
* MyMatrix            - class constructor
*
*                     - copy constructor
*
* ~Matrix             - class destructor
*
* copyInto            - copy the matrix and return the copy
*
*~~~~~~~~~~~~~~~~~~~~~~~~~
* my iterator:
*        in order to be able to iterete on both matrix kind I spand many hours to create
*        this (kind of) iterator.
*        I didn't succeed to make it a different class and couldn't spent more time on it.
*        The iterator use class member to store the current iterator(of the real iterator)
*        and current row and column. Therefore only 1 iteretor can use at a time (what make a problem
*        with operator+ as mention there)
*        every method have a boolen parameter "onlyNotZero" that is false by default.
*        In case it get "true", the sparse iteretor will run only on non-zero element.
*
* getFirst            - reset the iterator and return pointer to the first element
*
* hasNext             - return true iff there next element
*
* next                - increase the iterator and return the next element
*~~~~~~~~~~~~~~~~~~~~~~~~~
*
* getElem             - return the element in the (row, col) location
*
* getRow            - return the row
*
* setRow            - set the row
*
* getCol            - return the column
*
* setCol            - set the column
* ------------------------------------------------------------------------------------------------
*/
#include <string>
#include "Map.h"

using namespace std;

#ifndef _METRIX_H_
#define _METRIX_H_

/**
* This class is an abstract class represents the acuall matrix variable
* The matrix can be in 2 different way. regular and sparse.
* regular - more or equal to half of the values are nor zero, contained in a vector
* sparse - less than half are zero, contained in a map.
```

```
60  */
61  class Matrix
62  {
63  public:
64      /**
65      * class constructor
66      * set row and column
67      */
68      Matrix(unsigned int colSize, unsigned int rowSize);
69      /**
70      * copy constructor
71      */
72      Matrix(const Matrix& other);
73      /**
74      * class desstructor
75      */
76      virtual ~Matrix();
77
78      /**
79      * copy the matrix and return the copy
80      *
81      * pure virtual method
82      */
83      virtual Matrix* copyInto() = 0;
84      /**
85      * reset the iterator and return pointer to the first element
86      *
87      * pure virtual method
88      */
89      virtual double* getFirst(bool onlyNotZero = false) = 0;
90      /**
91      *    return true iff there next element
92      *
93      * pure virtual method
94      */
95      virtual bool hasNext(bool onlyNotZero = false) = 0;
96      /**
97      * increase the iterator and return the next element
98      *
99      * pure virtual method
100     */
101     virtual double* next(bool onlyNotZero = false) = 0;
102     /**
103     * return the element in the (row, col) location
104     *
105     * pure virtual method
106     */
107     virtual double getElem(const unsigned int row, const unsigned int col) const = 0;
108
109     /**
110     * return the row
111     */
112     unsigned int getRow() const
113     {
114         return _row;
115     }
116     /**
117     * set the row
118     */
119     void setRow(unsigned int other)
120     {
121         _row = other;
122     }
123     /**
124     * return the column
125     */
126     unsigned int getCol() const
127     {
```

general_bad_style(max/5){implementation should be done in cpp files not h files}

```
128             return _col;
129         }
130         /**
131         * set the column
132         */
133         void setCol(unsigned int other)
134         {
135             _col = other;
136         }
137 
138     private:
139         unsigned int _row;
140         unsigned int _col;
141 
142     };
143     #endif
```

5 Matrix.cpp

```cpp
/**
* Matrix.cpp
*
* ------------------------------------------------------------------------------------------------
* General:        This class is an abstract class represents the acuall matrix variable
*                 The matrix can be in 2 different way. regular and sparse.
*                 regular - more or equal to half of the values are nor zero, contained in a vector
*                 sparse - less than half are zero, contained in a map.
* ------------------------------------------------------------------------------------------------
*/
#include <iterator>
#include "Matrix.h"
#include "Vector.h"

using namespace std;

Matrix::Matrix(unsigned int colSize, unsigned int rowSize)
{
    setRow(rowSize);
    setCol(colSize);
}

Matrix::Matrix(const Matrix& other)
{
    setRow(other.getRow());
    setCol(other.getCol());
}

Matrix::~Matrix()
{
    _row = 0;
    _col = 0;
}
```

6 MyMatrix.h

```
/**
* MyMatrix.h
*
* ------------------------------------------------------------------------------------------------
* General:        This class represents a matrix and can operate thing such as M+N, M*N, M==n
*                 and more.
*                 The matrix can be in 2 different way. regular and sparse.
*                 regular - more or equal to half of the values are nor zero, contained in a vector
*                 sparse - less than half are zero, contained in a map.
*
* MyMatrix            - class constructor
*                       create a new colSizeXrowSize matrix from the values in arr
*
*                     - copy constructor
*
*                     - string constructor
*                       create a matrix from the givan string
*
* newMatrix             - create a colSizeXrowSize matrix from the values in arr
*
* ~MyMatrix             - class destructor
*
* operator=             - assign the other matrix into this
*
* operator<<        - ostream for the givan matrix
*
* operator+             - add this matrix with the other one and return the new matrix
*
* operator+=        - add this matrix the other one
*
* operator-(unary)    - return matrix with the opposite sign of every value in the matrix
*
* operator-             - decrease the other matrix from this one and return the new matrix
*
* operator-=        - and return the new matrix
*
* operator*             - multiply this matrix with the other one and return the new matrix
*
* operator*=        - multiply this matrix with the other one
*
* operator==        - return true iff all values are equals
*
* trace                 - return the trace of the matrix. 0 if the matrix is not a square
*                       trace - the sum of the diagonal values
*
* frobeniusNorm         - return the frobenius norm of the matrix
*                       frobenius norm - sum of the matrix values square
*
* isEmpty             - return true if all values are zero
*
* getMatrix             - return the matrix
*
* setMatrix             - set the matrix
*
* getZero             - return the zero counter
*
* setZero             - set the zero counter
*
* incZero             - increase the zero counter by 1
```

```
60  *
61  * decZero            - decrease the zero counter by 1
62  *
63  * getRow             - return the row
64  *
65  * setRow             - set the row
66  *
67  * getCol             - return the column
68  *
69  * setCol             - set the column
70  *
71  *
72  * switching between matrix kinds.
73  *    the way I implement the files, mostly for my own ease, most of the method are returning a new
74  *    matrix created *after* the operation goes. which mean the matrix already created in the right
75  *    form so there is no reason to switch anything.
76  * ------------------------------------------------------------------------------------------------
77  */
78  #include "Matrix.h"
79  #include "RegMatrix.h"
80  #include "SparseMatrix.h"
81  #include <string>
82  #include <iomanip>
83  #include "Map.h"
84
85  #ifndef EPSLON
86  #define EPSLON 0.000001
87  #endif
88  #ifndef PRECISON
89  #define PRECISON 6
90  #endif
91  #ifndef EMPTY_MAT
92  #define EMPTY_MAT "0\n"
93  #endif
94  #ifndef COMMA
95  #define COMMA " , "
96  #endif
97  #ifndef NEW_LINE
98  #define NEW_LINE "\n"
99  #endif
100
101 using namespace std;
102
103 #ifndef _MY_METRIX_H_
104 #define _MY_METRIX_H_
105 /**
106 * This class represents a matrix and can operate thing such as M + N, M*N, M == n
107 * and more.
108 * The matrix can be in 2 different way.regular and sparse.
109 * regular - more or equal to half of the values are nor zero, contained in a vector
110 * sparse - less than half are zero, contained in a map.
111 */
112 class MyMatrix
113 {
114 public:
115     /**
116     * class constructor
117     * create a new colSizeXrowSize matrix from the values in arr
118     */
119     MyMatrix(double arr[], unsigned int colSize, unsigned int rowSize);
120     /**
121     * copy constructor
122     */
123     MyMatrix(const MyMatrix& other);
124     /**
125     * string constructor
126     * create a matrix from the givan string
127     */
```

```
128         MyMatrix(const string matrix);
129         /**
130         * create a colSizeXrowSize matrix from the values in arr
131         * the method calculate the zero-valus sum then decide which matrix to create
132         */
133         void newMatrix(double arr[], unsigned int colSize, unsigned int rowSize);
134         /**
135         * class destructor
136         */
137         virtual ~MyMatrix();
138 
139         /**
140         * assign the other matrix into this
141         */
142         MyMatrix* operator=(const MyMatrix& other);
143 
144 
145         /**
146         * add this matrix with the other one and return the new matrix
147         */
148         MyMatrix operator+(const MyMatrix& other) const;
149 
150         /**
151         * add this matrix the other one
152         */
153         MyMatrix operator+=(const MyMatrix& other);
154 
155         /**
156         * return matrix with the opposite sign of every value in the matrix
157         */
158         MyMatrix operator-() const;
159 
160         /**
161         * decrease the other matrix from this one and return the new matrix
162         */
163         MyMatrix operator-(const MyMatrix& other) const;
164 
165         /**
166         * and return the new matrix
167         */
168         MyMatrix operator-=(const MyMatrix& other);
169 
170         /**
171         * multiply this matrix with the other one and return the new matrix
172         */
173         MyMatrix operator*(const MyMatrix& other) const;
174 
175         /**
176         * multiply this matrix with the other one
177         */
178         MyMatrix operator*=(const MyMatrix& other);
179 
180         /**
181         * return true iff all values are equals
182         */
183         bool operator==(const MyMatrix& other) const;
184 
185         /**
186         * return the trace of the matrix. 0 if the matrix is not a square
187         *
188         * trace - the sum of the diagonal values
189         */
190         double trace() const;
191 
192         /**
193         * return the frobenius norm of the matrix
194         *
195         * frobenius norm - sum of the matrix values square
```

```
196         */
197         double frobeniusNorm() const;
198 
199         /**
200         * ostream for the givan matrix
201         */
202         friend ostream& operator<<(ostream& os, const MyMatrix& matrix);
203 
204         /**
205         * return true if all values are zero
206         */
207         bool isEmpty() const;
208 
209         /**
210         * return the matrix
211         */
212         Matrix* getMatrix() const
213         {
214             return _matrix;
215         }
216         /**
217         * set the matrix
218         */
219         void setMatrix(Matrix* other)
220         {
221             _matrix = other;
222         }
223         /**
224         * return the zero counter
225         */
226         unsigned int getZero() const
227         {
228             return _zeroSum;
229         }
230         /**
231         * set the zero counter
232         */
233         void setZero(unsigned int other)
234         {
235             _zeroSum = other;
236         }
237         /**
238         * increase the zero counter by 1
239         */
240         void incZero()
241         {
242             _zeroSum++;
243         }
244         /**
245         * decrease the zero counter by 1
246         */
247         void decZero()
248         {
249             _zeroSum--;
250         }
251         /**
252         * return the row
253         */
254         unsigned int getRow() const
255         {
256             return _row;
257         }
258         /**
259         * set the row
260         */
261         void setRow(unsigned int other)
262         {
263             _row = other;
```

```
264         }
265         /**
266         * return the column
267         */
268         unsigned int getCol() const
269         {
270             return _col;
271         }
272         /**
273         * set the column
274         */
275         void setCol(unsigned int other)
276         {
277             _col = other;
278         }
279 
280 
281     private:
282         Matrix* _matrix;
283         unsigned int _zeroSum;
284         unsigned int _row;
285         unsigned int _col;
286 
287     };
288     #endif
```

7 MyMatrix.cpp

```cpp
/**
* MyMatrix.cpp
*
* ------------------------------------------------------------------------------------------------
* General:        This class represents a matrix and can operate thing such as M+N, M*N, M==n
*                 and more.
*                 The matrix can be in 2 different way. regular and sparse.
*                 regular - more or equal to half of the values are nor zero, contained in a vector
*                 sparse - less than half are zero, contained in a map.
* ------------------------------------------------------------------------------------------------
*/
#include <sstream>
#include <math.h>
#include <iterator>
#include "MyMatrix.h"
#include "Vector.h"


using namespace std;

MyMatrix::MyMatrix(double arr[], unsigned int colSize, unsigned int rowSize) :
                    _row(rowSize), _col(colSize)
{
    newMatrix(arr, colSize, rowSize);
}

MyMatrix::MyMatrix(const string other)
{
    string line, data;
    double* temp;
    char split_char = ',';
    unsigned int index;
    int row, col, curRow, curCol;
    istringstream split(other);

    row = 0;
    for (index = 0; index < other.length(); index++)
    {
        if (other[index] == '\n')
        {
            row++;
        }
    }
    col = 1;
    index = 0;
    while (other[index] != '\n')
    {
        index++;
        if (other[index] == ',')
        {
            col++;
        }
    }
    setRow(row);
    setCol(col);

    temp = new double[getRow()*getCol()];
    index = 0;
    curRow = 0;
```

```
60      while (getline(split, line))
61      {
62          curCol = 0;
63          istringstream split(line);
64          while (getline(split, data, split_char))
65          {
66              (*(temp + curCol*getRow() + curRow)) = stod(data);
67              curCol++;
68          }
69          curRow++;
70      }
71      newMatrix(temp, col, row);
72      delete[] temp;
73  }
74
75  MyMatrix::MyMatrix(const MyMatrix& other)
76  {
77      _row = other._matrix->getRow();
78      _col = other._matrix->getCol();
79      _zeroSum = other._zeroSum;
80      setMatrix(other._matrix->copyInto());
81  }
82
83  /**
84  * create a colSizeXrowSize matrix from the values in arr
85  * the method calculate the zero-valus sum then decide which matrix to create
86  */
87  void MyMatrix::newMatrix(double arr[], unsigned int colSize, unsigned int rowSize)
88  {
89      setZero(0);
90      int size = getRow()*getCol();
91      for (int i = 0; i < size; i++)
92      {
93          if (fabs(arr[i] - 0) < EPSLON)
94          {
95              incZero();
96          }
97      }
98      if ((float)getZero() / size <= 0.5)
99      {
100         setMatrix(new RegMatrix(arr, getCol(), getRow()));
101     }
102     else
103     {
104         setMatrix(new SparseMatrix(arr, getCol(), getRow()));
105     }
106 }
107
108 MyMatrix::~MyMatrix()
109 {
110     _row = 0;
111     _col = 0;
112     delete _matrix;
113 }
114 /**
115 * assign the other matrix into this
116 * the assignment create a new matrix of the type of other (no need to switch)
117 */
118 MyMatrix* MyMatrix::operator=(const MyMatrix& other)
119 {
120     if (this == &other)
121     {
122         return this;
123     }
124     setRow(other.getRow());
125     setCol(other.getCol());
126     delete _matrix;
127     setMatrix(other.getMatrix()->copyInto());
```

```
128     setZero(other.getZero());
129     return this;
130 }
131
132 /**
133 * add this matrix with the other one and return the new matrix
134 *
135 * the method create an array of the new matrix to be then create a new matrix from it.
136 *
137 * due some implementetion difficulte i did it without the iterator
138 * (i couldn't run my iterator in case both matrix are acually the same)
139 */
140 MyMatrix MyMatrix::operator+(const MyMatrix& other) const
141 {
142     //in case one of the matrix is zero
143     if (isEmpty())
144     {
145         return other;
146     }
147     if (other.isEmpty())
148     {
149         return *this;
150     }
151
152     double* tmpArr = new double[getRow() * getCol()]();
153
154     for (unsigned int row = 0; row < getRow(); row++)
155     {
156         for (unsigned int col = 0; col < getCol(); col++)
157         {
158             tmpArr[getRow()*col + row] =
159                 this->_matrix->getElem(row, col) + other._matrix->getElem(row, col);
160         }
161     }
162     MyMatrix temp(tmpArr, getCol(), getRow());
163     delete[] tmpArr;
164     return temp;
165
166 }
167
168 MyMatrix MyMatrix::operator+=(const MyMatrix& other)
169 {
170     *this = *this + other;
171     return *this;
172 }
173
174 /**
175 * decrease the other matrix from this one and return the new matrix
176 *
177 * the method create a new matrix then change all sign.
178 *
179 * using my iterator method
180 * in case of sparse matrix the iterator run only on non-zero values
181 */
182 MyMatrix MyMatrix::operator-() const
183 {
184     MyMatrix temp(*this);
185     double* slot = temp.getMatrix()->getFirst(true);
186     if (fabs(*slot) > EPSLON)
187     {
188         *slot = -*slot;
189     }
190     while (temp.getMatrix()->hasNext(true))
191     {
192         slot = temp.getMatrix()->next(true);
193         if (fabs(*slot) > EPSLON)
194         {
195             *slot = -*slot;
```

Note that you handle reg and sparse matrix exactly in the same manner - this is good from the polymorphic point of view, but maybe in some operations you could've used the sparseness property to perform operations more efficiently

badSparseUsage{If for example you add two 100*100 matrices containing only 2 non zero elements, isn't this a waste to go over all the elements?}

```
196         }
197     }
198     return temp;
199 }
200 
201 MyMatrix MyMatrix::operator-(const MyMatrix& other) const
202 {
203     MyMatrix temp = -other;
204     return *this + temp;
205 }
206 
207 MyMatrix MyMatrix::operator-=(const MyMatrix& other)
208 {
209     *this = *this - other;
210     return *this;
211 }
212 
213 /**
214 * multiply this matrix with the other one and return the new matrix
215 *
216 * the method create an array of the new matrix to be then create a new matrix from it.
217 *
218 * due some implementetion difficulte i did it without the iterator
219 */
220 MyMatrix MyMatrix::operator*(const MyMatrix& other) const
221 {
222     unsigned int tmpRow, tmpCol;
223     double* tmpArr;
224     tmpRow = this->getRow();
225     tmpCol = other.getCol();
226     tmpArr = new double[tmpRow * tmpCol]();
227 
228     for (unsigned int row = 0; row < tmpRow; row++)
229     {
230         for (unsigned int col = 0; col < tmpCol; col++)
231         {
232             for (unsigned int i = 0; i < this->getCol(); i++)
233             {
234                 tmpArr[tmpRow*col + row] +=
235                     this->getMatrix()->getElem(row, i)*other.getMatrix()->getElem(i, col);
236             }
237         }
238     }
239     MyMatrix temp(tmpArr, tmpRow, tmpCol);
240     delete[] tmpArr;
241     return temp;
242 }
243 
244 MyMatrix MyMatrix::operator*=(const MyMatrix& other)
245 {
246     *this = (*this) * other;
247     return *this;
248 }
249 
250 /**
251 * return true iff all values are equals
252 *
253 * using my iterator method
254 * in case of sparse matrix the iterator run only on non-zero values
255 */
256 bool MyMatrix::operator==(const MyMatrix& other) const
257 {
258     //check some fast way to decide equality
259     if (this == &other)
260     {
261         return true;
262     }
263     if ((_zeroSum != other._zeroSum) || (getRow() != other.getRow()) || (getCol() != other.getCol()))
```

```
264         {
265             return false;
266         }
267 
268         double* lhs = this->getMatrix()->getFirst(true);
269         double* rhs = other.getMatrix()->getFirst(true);
270         if (fabs(*lhs - *rhs) > EPSLON)
271         {
272             return false;
273         }
274         while (this->getMatrix()->hasNext(true) && this->getMatrix()->hasNext(true))
275         {
276             lhs = this->getMatrix()->next(true);
277             rhs = other.getMatrix()->next(true);
278             if (fabs(*lhs - *rhs) > EPSLON)
279             {
280                 return false;
281             }
282         }
283         return true;
284     }
285 
286     double MyMatrix::trace() const
287     {
288         double sum = 0;
289         if (getRow() != getCol())
290         {
291             return sum;
292         }
293         for (unsigned int i = 0; i < getRow(); i++)
294         {
295             sum += (getMatrix()->getElem(i, i));
296         }
297         return sum;
298     }
299 
300     /**
301     * return the frobenius norm of the matrix
302     *
303     * frobenius norm - sum of the matrix values square
304     *
305     * using my iterator method
306     * in case of sparse matrix the iterator run only on non-zero values
307     */
308     double MyMatrix::frobeniusNorm() const
309     {
310         if (_zeroSum == getRow() * getCol())
311         {
312             return 0;
313         }
314         double* data = getMatrix()->getFirst(true);
315         double sum = (*data)*(*data);
316         while (getMatrix()->hasNext(true))
317         {
318             data = getMatrix()->next(true);
319             sum += (*data)*(*data);
320         }
321         return sum;
322     }
323 
324     /**
325     * ostream for the givan matrix
326     *
327     * using my iterator method
328     */
329     ostream& operator<<(ostream& os, const MyMatrix& matrix)
330     {
331         if (matrix.isEmpty())
```

```
332     {
333         os << EMPTY_MAT;
334         return os;
335     }
336     double* iter = matrix.getMatrix()->getFirst();
337     for (unsigned int r = 0; r < matrix.getMatrix()->getRow(); r++)
338     {
339         os << setprecision(PRECISON + (int)log10(*iter)) << *iter;
340         for (unsigned int c = 1; c < matrix.getMatrix()->getCol(); c++)
341         {
342             os << COMMA;
343             iter = matrix.getMatrix()->next();
344             os << setprecision(PRECISON + (int)log10(*iter)) << *iter;
345         }
346         iter = matrix.getMatrix()->next();
347         os << NEW_LINE;
348     }
349     return os;
350 }
351
352 bool MyMatrix::isEmpty() const
353 {
354     return (_zeroSum == getRow() * getCol());
355 }
```

8 RegMatrix.h

```
/**
* RegMatrix.h
*
* ------------------------------------------------------------------------------------------------ -
* General:        This class  represents regular matrix
*                the matrix contain more or equal than half zero, stored in a vector.
*
* MyMatrix            - class constructor
*                      create a new colSizeXrowSize sparse matrix from the values in arr
*
*                    - copy constructor
*
* ~Matrix            - class destructor
*
* copyInto            - copy the matrix and return the copy
*
*~~~~~~~~~~~~~~~~~~~~~~~~~
* my iterator:
*        in order to be able to iterete on both matrix kind I spand many hours to create
*        this (kind of) iterator.
*        I didn't succeed to make it a different class and couldn't spent more time on it.
*        The iterator use class member to store the current iterator(of the real iterator)
*        and current row and column. Therefore only 1 iteretor can use at a time (what make a prablem
*        with operator+ as mention there)
*        every method have a boolen parameter "onlyNotZero" that is false by default.
*        In case it get "true", the sparse iteretor will run only on non-zero element.
*
* getFirst            - reset the iterator and return pointer to the first element
*
* hasNext            - return true iff there next element
*
* next                - increase the iterator and return the next element
*~~~~~~~~~~~~~~~~~~~~~~~~~
*
* getElem            - return the element in the (row, col) location
*
* ------------------------------------------------------------------------------------------------
*/
#include "Matrix.h"
#include "Vector.h"
#include "Map.h"


using namespace std;

#ifndef _REG_METRIX_H_
#define _REG_METRIX_H_

/**
* This class  represents regular matrix
* the matrix contain more or equal than half zero, stored in a vector.
*/
class RegMatrix : public Matrix
{
public:
    /**
    * class constructor
    * create a new colSizeXrowSize regular matrix from the values in arr
    */
```

```
60      RegMatrix(double arr[], unsigned int colSize, unsigned int rowSize);
61      /**
62      * copy constructor
63      */
64      RegMatrix(const RegMatrix& other);
65      /**
66      * class desstructor
67      */
68      ~RegMatrix();
69
70      /**
71      * copy the matrix and return the copy
72      */
73      RegMatrix* copyInto();
74      /**
75      * reset the iterator and return pointer to the first element
76      */
77      virtual double* getFirst(bool onlyNotZero = false);
78      /**
79      *    return true iff there next element
80      */
81      virtual bool hasNext(bool onlyNotZero = false);
82      /**
83      * increase the iterator and return the next element
84      */
85      virtual double* next(bool onlyNotZero = false);
86      /**
87      * return the element in the (row, col) location
88      */
89      double getElem(const unsigned int row, const unsigned int col) const;
90
91  private:
92      Vector _matrix;
93
94      //for the iterator
95      Vector::iterator iter;
96  };
97  #endif
```

9 RegMatrix.cpp

```cpp
/**
* RegMatrix.cpp
*
* ------------------------------------------------------------------------------------------------
* General:        This class  represents regular matrix
*                 the matrix contain more or equal than half zero, stored in a vector.
* ------------------------------------------------------------------------------------------------
*/
#include <iterator>
#include "Vector.h"
#include "RegMatrix.h"

using namespace std;

RegMatrix::RegMatrix(double arr[], unsigned int colSize, unsigned int rowSize):
Matrix(colSize, rowSize)
{
    for (unsigned int r = 0; r < rowSize; r++)
    {
        for (unsigned int c = 0; c < colSize; c++)
        {
            _matrix.push_back(*(arr + c*rowSize + r));
        }
    }
}

RegMatrix::RegMatrix(const RegMatrix& other) : Matrix(other)
{
    setRow(other.getRow());
    setCol(other.getCol());
    _matrix = other._matrix;
}

RegMatrix::~RegMatrix()
{
    _matrix.clear();
}
RegMatrix* RegMatrix::copyInto()
{
    RegMatrix* temp = new RegMatrix(*this);
    return temp;
}

double RegMatrix::getElem(const unsigned int row, const unsigned int col) const
{
    return _matrix[row*getCol() + col];
}
double* RegMatrix::getFirst(bool onlyNotZero)
{
    iter = _matrix.begin();
    return &(*iter++);
}

bool RegMatrix::hasNext(bool onlyNotZero)
{
    return iter != _matrix.end();
}
double* RegMatrix::next(bool onlyNotZero)
{
```

```
60        return &(*iter++);
61    }
```

10 SparseMatrix.h

```
/**
* SparseMatrix.h
*
* ------------------------------------------------------------------------------------------------
* General:        This class  represents sparse matrix
*                the matrix contain less than half are zero, stored in a map.
*
* MyMatrix            - class constructor
*                      create a new colSizeXrowSize sparse matrix from the values in arr
*
*                    - copy constructor
*
* ~Matrix            - class destructor
*
* copyInto            - copy the matrix and return the copy
*
*~~~~~~~~~~~~~~~~~~~~~~~~~
* my iterator:
*        in order to be able to iterete on both matrix kind I spand many hours to create
*        this (kind of) iterator.
*        I didn't succeed to make it a different class and couldn't spent more time on it.
*        The iterator use class member to store the current iterator(of the real iterator)
*        and current row and column. Therefore only 1 iteretor can use at a time (what make a prablem
*        with operator+ as mention there)
*        every method have a boolen parameter "onlyNotZero" that is false by default.
*        In case it get "true", the sparse iteretor will run only on non-zero element.
*
* getFirst            - reset the iterator and return pointer to the first element
*
* hasNext            - return true iff there next element
*
* next                - increase the iterator and return the next element
*~~~~~~~~~~~~~~~~~~~~~~~~~
*
* getElem            - return the element in the (row, col) location
*
* ------------------------------------------------------------------------------------------------
*/
#include "Matrix.h"
#include "Map.h"

#ifndef EPSLON
#define EPSLON 0.000001
#endif

using namespace std;

#ifndef _SPARSE_METRIX_H_
#define _SPARSE_METRIX_H_

/**
* This class  represents sparse matrix
* the matrix contain less than half are zero, stored in a map.
*/
class SparseMatrix : public Matrix
{
public:
    /**
    * class constructor
```

```
60      * create a new colSizeXrowSize sparse matrix from the values in arr
61      */
62      SparseMatrix(double arr[], unsigned int colSize, unsigned int rowSize);
63      /**
64      * copy constructor
65      */
66      SparseMatrix(const SparseMatrix& other);
67      /**
68      * class desstructor
69      */
70      virtual ~SparseMatrix();
71      /**
72      * copy the matrix and return the copy
73      */
74      SparseMatrix* copyInto();
75      /**
76      * reset the iterator and return pointer to the first element
77      */
78      virtual double* getFirst(bool onlyNotZero = false);
79      /**
80      *    return true iff there next element
81      */
82      virtual bool hasNext(bool onlyNotZero = false);
83      /**
84      * increase the iterator and return the next element
85      */
86      virtual double* next(bool onlyNotZero = false);
87      /**
88      * return the element in the (row, col) location
89      */
90      double getElem(const unsigned int row, const unsigned int col) const;
91
92  private:
93      Map _matrix;
94
95      //for the iterator
96      Map::iterator iter;
97      unsigned int iterRow;
98      unsigned int iterCol;
99
100     // double pointer to 0. use for the iterator for element of 0
101     double empty = 0;
102     double* emptySlot = &empty;
103 };
104 #endif
```

11 SparseMatrix.cpp

```cpp
/**
* SparseMatrix.cpp
*
* ------------------------------------------------------------------------------------------------
* General:        This class  represents sparse matrix
*                 the matrix contain less than half are zero, stored in a map.
* ------------------------------------------------------------------------------------------------
*/
#include <math.h>
#include "Map.h"
#include "SparseMatrix.h"

using namespace std;

SparseMatrix::SparseMatrix(double arr[], unsigned int colSize, unsigned int rowSize) :
Matrix(colSize, rowSize)
{
    Pair key;
    double val;
    for (unsigned int c = 0; c < colSize; c++)
    {
        key.second = c;
        for (unsigned int r = 0; r < rowSize; r++)
        {
            key.first = r;
            val = *arr++;
            if (fabs(val - 0) > EPSLON)
            {
                _matrix[key] = val;
            }
        }
    }
}

SparseMatrix::SparseMatrix(const SparseMatrix& other) : Matrix(other)
{
    setRow(other.getRow());
    setCol(other.getCol());
    _matrix = other._matrix;
}

SparseMatrix* SparseMatrix::copyInto()
{
    SparseMatrix* temp = new SparseMatrix(*this);
    return temp;
}

SparseMatrix::~SparseMatrix()
{
    _matrix.clear();
}

double* SparseMatrix::getFirst(bool onlyNotZero)
{
    iter = _matrix.begin();
    iterRow = 0;
    iterCol = 0;

    //in case of the zer matrix
```

```
60      if (iter == _matrix.end())
61      {
62          return emptySlot;
63      }
64
65      if ((iter->first.first == iterRow && iter->first.second == iterCol) || onlyNotZero)
66      {
67          return &((*iter++).second);
68      }
69      else
70      {
71          return emptySlot;
72      }
73  }
74
75  bool SparseMatrix::hasNext(bool onlyNotZero)
76  {
77      if (onlyNotZero)
78      {
79          return iter != _matrix.end();
80      }
81      else
82      {
83          return (iterRow != getRow() - 1) || iterCol != getCol() - 1;
84      }
85  }
86
87  double* SparseMatrix::next(bool onlyNotZero)
88  {
89
90      if (iterCol == getCol() - 1)
91      {
92          iterCol = 0;
93          iterRow++;
94      }
95      else
96      {
97          iterCol++;
98      }
99      if ((iter->first.first == iterRow && iter->first.second == iterCol && iter != _matrix.end())
100         || onlyNotZero)
101     {
102         return &(iter++)->second;
103     }
104     else
105     {
106         return emptySlot;
107     }
108 }
109
110 double SparseMatrix::getElem(const unsigned int row, const unsigned int col) const
111 {
112     Pair key(row, col);
113     if (_matrix.find(key) == _matrix.end())
114     {
115         return 0;
116     }
117     return _matrix.at(key);
118 }
```

12 Vector.h

```
1  /**
2   * A vector that contains doubles
3   * This is simply a wrapper, or a typedef of the template vector STL
4   * Support iterators, assingment, and more methods.
5   */
6  #ifndef _VECTOR_H_
7  #define _VECTOR_H_
8
9  #include <vector>
10 typedef std::vector<double> Vector;
11 typedef std::vector<std::vector<double> > Vector2D;
12
13 #endif // _VECTOR_H_
```