```
/*
 * Vector_Line.h
 *
 *  Created on: 27 nov. 2015
 *      Author: aurelien.bach
 */

#ifndef VECTOR_LINE_H_
#define VECTOR_LINE_H_

#include "Matrix_Full.h"

namespace tp2{

class Vector_Line : public Matrix_Full{
public:
      Vector_Line(int n);
      int get_size()const{return get_col();}

      double & operator[](int i);       // ecriture
      double operator[]( int i ) const;  // lecture

      double norm ( int p=2 ) const;
      friend double operator* ( const Vector_Line & v1 , const Vector_Line &
v2 );
};

}// namespace tp2

#endif /* VECTOR_LINE_H_ */
```

```
/*
 * Matrix_Sparse.h
 *
 *  Created on: 27 nov. 2015
 *      Author: Utilisateur
 */

#ifndef MATRIX_SPARSE_H_
#define MATRIX_SPARSE_H_

#include <iostream>
#include <vector>
#include "BMatrix.h"
#include "Matrix_Sparse_Data.h"

namespace tp2{

class Vector_Line;
class Vector_Column;

class Matrix_Sparse : public BMatrix{
protected:
      std::vector<Matrix_Sparse_Data> data_;
public:
      Matrix_Sparse( int lin , int col );
      virtual ~Matrix_Sparse();
      Matrix_Sparse( const Matrix_Sparse & m );
      Matrix_Sparse & operator= ( const Matrix_Sparse & m );

      virtual double & operator()( int i , int j ); // ecriture
      virtual double operator()( int i , int j ) const;   // lecture


      Matrix_Sparse & operator+=( const Matrix_Sparse & m );
      Matrix_Sparse & operator-=( const Matrix_Sparse & m );


      Matrix_Sparse transp() const;

      friend Matrix_Sparse operator+ ( const Matrix_Sparse & m1 , const
Matrix_Sparse & m2 );
      friend Matrix_Sparse operator- ( const Matrix_Sparse & m1 , const
Matrix_Sparse & m2 );
      friend Matrix_Sparse operator* ( const Matrix_Sparse & m1 , const
Matrix_Sparse & m2 );
      friend Matrix_Sparse operator* ( double p , const Matrix_Sparse & m );

      friend Vector_Line operator* ( const Vector_Line & v , const Matrix_Sparse
& m );
      friend Vector_Column operator* ( const Matrix_Sparse & m , const
Vector_Column & v );

      friend std::ostream & operator<< (std::ostream & st, const Matrix_Sparse
&m );
};


}//namespace tp2



#endif /* MATRIX_SPARSE_H_ */
```

```
/*
 * BMatrix.h
 *
 *  Created on: 27 nov. 2015
 *      Author: aurelien.bach
 */

#ifndef BMATRIX_H_
#define BMATRIX_H_

namespace tp2{

class BMatrix{
protected:
	int lin_;
	int col_;
	BMatrix(int lin, int col){
		if ( lin < 0 || col < 0 ) throw "taille < 0";
		else {
			lin_ = lin;
			col_ = col;
		}
	}
	virtual ~BMatrix(){}
public:
	int get_lin() const {return lin_;}
	int get_col() const {return col_;}
	bool same_size( const BMatrix & m )const{return (m.col_ == col_ && m.lin_
== lin_);}
	virtual double & operator()( int i , int j )=0;		// ecriture
	virtual double operator()( int i , int j ) const=0;	// lecture
};


}//namespace tp2

#endif /* BMATRIX_H_ */
```

```
/*
 * Matrix_Sparse_2.h
 *
 *  Created on: 18 déc. 2015
 *      Author: Utilisateur
 */

#ifndef MATRIX_SPARSE_2_H_
#define MATRIX_SPARSE_2_H_

#include <iostream>
#include <vector>
#include "BMatrix.h"

namespace tp2{

class Vector_Line;
class Vector_Column;

class Matrix_Sparse_2: public BMatrix{
protected:
      std::vector<double> val_;
      std::vector<int> idx_;
      int* start_;  // taille = lin + 1
public:
      Matrix_Sparse_2( int lin , int col );
      virtual ~Matrix_Sparse_2();
      Matrix_Sparse_2( const Matrix_Sparse_2 & m );
      Matrix_Sparse_2 & operator= ( const Matrix_Sparse_2 & m );

      virtual double & operator()( int i , int j );       // ecriture
      virtual double operator()( int i , int j ) const;   // lecture

      //Matrix_Sparse_2 & operator+=( const Matrix_Sparse & m );
      //Matrix_Sparse_2 & operator-=( const Matrix_Sparse & m );

      /*

      Matrix_Sparse_2 transp() const;

      friend Matrix_Sparse_2 operator+ ( const Matrix_Sparse_2 & m1 , const
Matrix_Sparse_2 & m2 );
      friend Matrix_Sparse_2 operator- ( const Matrix_Sparse_2 & m1 , const
Matrix_Sparse_2 & m2 );
      friend Matrix_Sparse_2 operator* ( const Matrix_Sparse_2 & m1 , const
Matrix_Sparse_2 & m2 );
      friend Matrix_Sparse_2 operator* ( double p , const Matrix_Sparse_2 & m );

      friend Vector_Line operator* ( const Vector_Line & v , const
Matrix_Sparse_2 & m );
      friend Vector_Column operator* ( const Matrix_Sparse_2 & m , const
Vector_Column & v );
*/
      friend std::ostream & operator<< (std::ostream & st, const Matrix_Sparse_2
&m );

};

}// namespace tp2

#endif /* MATRIX_SPARSE_2_H_ */
```

```
/*
 * Matrix_Full.h
 *
 *  Created on: 27 nov. 2015
 *      Author: aurelien.bach
 */

#ifndef MATRIX_FULL_H_
#define MATRIX_FULL_H_

#include "BMatrix.h"
#include <iostream>

namespace tp2{

class Vector_Line;
class Vector_Column;

class Matrix_Full : public BMatrix {
protected:
	double* data_;
public:
	Matrix_Full( int lin , int col );
	virtual ~Matrix_Full();
	Matrix_Full( const Matrix_Full & m );
	Matrix_Full & operator= ( const Matrix_Full & m );

	virtual double & operator()( int i , int j ); // ecriture
	virtual double operator()( int i , int j ) const;   // lecture

	Matrix_Full & operator+=( const Matrix_Full & m );
	Matrix_Full & operator-=( const Matrix_Full & m );

	Matrix_Full transp() const;

	friend Matrix_Full operator+ ( const Matrix_Full & m1 , const Matrix_Full
& m2 );
	friend Matrix_Full operator- ( const Matrix_Full & m1 , const Matrix_Full
& m2 );
	friend Matrix_Full operator* ( const Matrix_Full & m1 , const Matrix_Full
& m2 );
	friend Matrix_Full operator* ( double p , const Matrix_Full & m );

	friend Vector_Line operator* ( const Vector_Line & v , const Matrix_Full &
m );
	friend Vector_Column operator* ( const Matrix_Full & m , const
Vector_Column & v );

	friend std::ostream & operator<< (std::ostream & st, const Matrix_Full
&m );
};

}// namespace tp2



#endif /* MATRIX_FULL_H_ */
```

```
/*
 * Matrix_Sparse_Data.h
 *
 *  Created on: 27 nov. 2015
 *      Author: Utilisateur
 */

#ifndef MATRIX_SPARSE_DATA_H_
#define MATRIX_SPARSE_DATA_H_

namespace tp2{

class Matrix_Sparse_Data{
public:
      int i_;
      int j_;
      double val_;
public:
      Matrix_Sparse_Data( int i , int j , double val ):i_(i),j_(j),val_(val){}
      int get_i() const{return i_;}
      int get_j() const{return j_;}
      double get_val() const{return val_;}
};

}// namespace tp2

#endif /* MATRIX_SPARSE_DATA_H_ */
```

```
/*
 * Vector_Column.h
 *
 *  Created on: 27 nov. 2015
 *      Author: aurelien.bach
 */

#ifndef VECTOR_COLUMN_H_
#define VECTOR_COLUMN_H_

#include "Matrix_Full.h"

namespace tp2{

class Vector_Column : public Matrix_Full{
public:
      Vector_Column(int n);
      int get_size()const{return get_lin();}

      double & operator[](int i);       // ecriture
      double operator[]( int i ) const;  // lecture

      double norm ( int p=2 ) const;
      friend double operator* ( const Vector_Column & v1 , const Vector_Column &
v2 );
};

}// namespace tp2

#endif /* VECTOR_COLUMN_H_ */
```

```
/*
 * Matrix_Sparse_2.cpp
 *
 *  Created on: 18 déc. 2015
 *      Author: Utilisateur
 */

#include "Matrix_Sparse_2.h"
#include "Vector_Line.h"
#include "Vector_Column.h"

namespace tp2{

Matrix_Sparse_2::Matrix_Sparse_2(int lin,int col):BMatrix(lin,col){
        start_ = new int[lin+1];
        if ( !start_ ){
                throw "echec allocation";
        }
        for ( int i = 0 ; i < lin+1 ; i++ ){
                start_[i] = 0;
        }
}

Matrix_Sparse_2::~Matrix_Sparse_2(){
        delete[] start_;
}

Matrix_Sparse_2::Matrix_Sparse_2( const Matrix_Sparse_2 & m
):BMatrix(m.lin_,m.col_){
        start_ = new int[m.lin_+1];
        if ( !start_ ) throw "pb allocation start_";
        for ( int k = 0 ; k < (int)m.val_.size() ; k++ ){
                val_.push_back( m.val_[k] );
                idx_.push_back( m.idx_[k] );
        }
        for ( int i = 0 ; i < m.lin_ + 1 ; i++ ){
                start_[i] = m.start_[i];
        }
}

Matrix_Sparse_2 & Matrix_Sparse_2::operator= ( const Matrix_Sparse_2 & m ){
        if( &m != this ){
                Matrix_Sparse_2 tmp = m;

                std::vector<double> tval = tmp.val_;
                tmp.val_ = val_;
                val_ = tval;

                std::vector<int> tidx = tmp.idx_;
                tmp.idx_ = idx_;
                idx_ = tidx;

                int* st = tmp.start_;
                tmp.start_ = start_;
                start_ = st;

                lin_ = m.lin_;
                col_ = m.col_;
        }
        return *this;
}

// ecriture
double & Matrix_Sparse_2::operator()( int i , int j ){
        int k = start_[i];
        while( k < start_[i+1] && idx_[k] != j ){
                k++;
        }
```

```
        if ( k == start_[i+1] ){
                val_.insert( val_.begin()+k , 0 );
                idx_.insert( idx_.begin()+k , j );
                for ( int p = i+1 ; p < lin_+1 ; p++ ){
                        start_[p]++;
                }
        }
        return val_[k];
}

// lecture
double Matrix_Sparse_2::operator()( int i , int j )const{
        double res;
        if ( start_[i+1] - start_[i] == 0 ){
                res = 0;
        } else {
                int k = start_[i];
                while( k < start_[i+1] && idx_[k] != j ){
                        k++;
                }
                if ( k == start_[i+1] ){
                        res = 0;
                } else {
                        res = val_[k];
                }
        }
        return res;
}

std::ostream & operator<< (std::ostream & st, const Matrix_Sparse_2 &m ){
        for ( int i = 0 ; i < m.get_lin() ; i++ ){
                for ( int j = 0 ; j < m.get_col() ; j++ ){
                        st << m(i,j) << " ";
                }
                st << "\n";
        }
        st << "\n";
        return st;
}

}// namespace tp2
```