Implement the following inherited classes which represent square, lower triangular, and upper triangular matrices from base class `Matrix`. The inheritance hierarchy is shown in Figure 1.

Complete the following tasks:

- **Task 1**: *Modifying the base class* `Matrix`

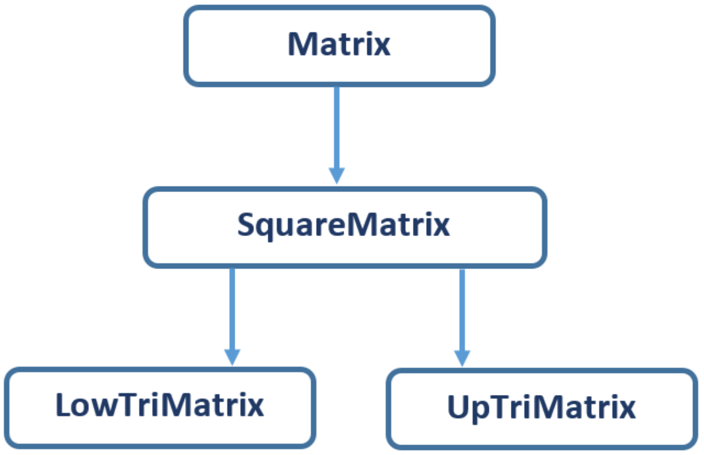

Figure 1: Matrix classes inheritance hierarchy

```
/*****************************************************
* class Matrix:
*     1. private --> protected
*
*     2. virtual member methods, which allows polymorphism
*       virtual ~Matrix()
*       virtual void zeros() const
*       virtual void ones() const
*       virtual void random() const
*
*    3. additional methods
*       bool isSquare() const
*********************************************************/
```

Listing 1: class Matrix

- **Task 2**: *Implementing all methods for the derived class* `SquareMatrix` *whose structure is given in details below.*

```
/*****************************************************
* class SquareMatrix:
*     1. inherited type: pulic from the base class Matrix
*
*     2. int size = numRows = numCols (protected)
*
*     3. constructors:
*       SquareMatrix(size)
*       SquareMatrix(size,val)
```

```
 *       SquareMatrix(const Matrix& A)
 *       SquareMatrix(const SquareMatrix& A)
 *
 *     4. virtual methods
 *       virtual ~SquareMatrix()
 *       virtual double det() const
 *
 *     5. overloaded assignment operators
 *       SquareMatrix& operator=(const Matrix& mat);
 *       SquareMatrix& operator=(const SquareMatrix& mat);
 *
 *     6. additional or modified methods:  *
 *       int getSize() const
 *       bool isLowTri() const
 *       bool isUpTri() const
 *       SquareMatrix cofactor(const int& p, const int& q) const;
 *       SquareMatrix adjoint() const;
 *       SquareMatrix inverse() const;
 ************************************************************/
```

Listing 2: class SquareMatrix

- **Task 3**: *Based on classes* ***Matrix*** *and* ***SquareMatrix****, develop derived classes* ***LowTriMatrix*** *and* ***UpTriMatrix****.*

  (i) What are the additional methods?

  (ii) What methods need overriding through polymorphism?

Note the the determinant of a triangular matrix T can be simplified by

$$det(T) = \prod_{k=0}^{N-1} T_{kk}$$

Use the following `main()` function to test your implementation

```
#include <iostream>
#include "vector.h"
#include "matrix.h"
#include "smatrix.h"
#include "ltmatrix.h"
#include "utmatrix.h"
using namespace std;

int main()
{
  // create some random square matrices
  SquareMatrix A(5);
  A.random();
  SquareMatrix B(5);
  B.random();
  SquareMatrix C(5);
  C.random();

  cout << "Square matrix A: " << endl;
  A.print();
  cout << "Square matrix B: " << endl;
  B.print();
  cout << "Square matrix C: " << endl;
  C.print();

  // carry out some linear algebra calculations
```

```
    SquareMatrix D(5, 0.0);
    D = ++(-A + B*C);
    cout << "D␣=␣++(␣-A␣+␣B*C␣):␣" << endl;
    D.print();

    // check that whether D is invertible
    double detD = D.det();
    cout << "det(D)␣=␣" << detD << endl;

    // vector b
    Vector b(5);
    b.random();
    cout << "Vector␣b:␣" << endl;
    b.print();

    // solve the linear system D*x = b
    // for the unknown vector x
    Vector x = D.inverse() * b;
    cout << "Solution␣x␣=␣Dinverse␣*␣b:␣" << endl;
    x.print();
    cout << "Error␣||␣D*x␣-␣b␣||_2␣=␣"
      << (D * x - b).norm(2) << endl;

    //======== lowtri matrices ===========
    MatrixDoubleLowTri L1(5);
    L1.random(ONE, TEN);
    L1.print();

    // copy constructors
    //MatrixDoubleLowTri L2(S1);  // should return error
    //MatrixDoubleLowTri L2(B); // should return error
    MatrixDoubleLowTri L2(L1);
    L2.print();

    // operators: the outcome of these operators must be an lowtri matrix
    // otherwise return run-time error
    MatrixDoubleLowTri L3(5);
    L3.zeros();
    L3 = L1 + L2;
    //L3 = L1 + S1; // should return errors
    L3.print();
  }
```

Listing 3: int main()