**Homework 4: ADT design and implementation.**

**The main goal of this homework** is three-fold:

1. applying the problem-solving process to design and implement two ADTs to represent two types of objects;
2. implementing the ADTs using separate compilation;
3. practicing to use several common ADT features--const, explicit, and operator overloading.

**Submission**: this homework contains two parts, corresponding to two ADTs.  For each part, please first compress your three files (.h and two .cpp) in one zip file on your local computer.  Then submit these two zip files.

**Problem I --Polynomial ADT. (50 points).**

Use separate compilation to implement a polynomial ADT to manipulate the following type of polynomials: (1)  involving a single variable x,  e.g., p = 5.1 \* x^10 + 7.2 \* x^4 - 11.0 (where the number after the ^ symbol is the exponent/degree of a term); (2) the degree of each term is a non-negative integer and doesn't exceed 200; and (3) the coefficient of each term is a floating number. Please identify a proper data representation schema to store such polynomials and hide such data from external users. Additionally, your ADT is required to include the following member functions:

1. A default constructor, which initializes the polynomial to 0.0
2. A constructor with one integer parameter, which indicates the highest degree allowed of a given  polynomial.
3. A read-only accessor *degree()* that returns the degree of a polynomial, which is defined as the highest power of any term contained in the polynomial with a nonzero coefficient. For instance, the degree of the above underlined polynomial is 10.
4. A read-only accessor *getCoeff(int power)* that returns the coefficient of the term x^(*power*).
5. One mutator that interacts with the end user to to obtain a polynomial from the keyboard. For example, if a polynomial contains three terms, your method will ask the end user to type in the three terms one by one. For each term, you are going to prompt the user to type its coefficient and then degree.
6. A mutator *setCoeff(int power, double newCoefficient)* that sets the coefficient of  the term x^(*power*) to *newCoefficient*.
7. An overloaded division operator (/) as a member function. This operator divides a polynomial by a scalar variable. Also, this operator is not allowed to change the calling object.  For instance, let p = 5.1 \* x^10 + 7.2 \* x^4 - 11.0,  p/2.0 = 2.55 \* x^10 + 3.6 \* x^4 - 5.50, but p will remain unchanged.
8. An overloaded negation operator (-) as a member function. This operator will negate the coefficient of each term in a polynomial. For instance, given p = 5.1 \* x^10 + 7.2 \* x^4 - 11.0, -p will not only change p to -5.1 \* x^10 - 7.2 \* x^4 + 11.0but also returns this negated polynomial.
9. An overloaded addition (+) operator as a member function that sums up two polynomials.  This operator is not allowed to change either of the two polynomials participating in the addition and returns the sum as another polynomial.  For instance, let p1, p2, p3 be three polynomials, the statement "p3=p1+p2" will not change p1 or p2 and will save the sum to p3.
10. Finally, overload the “put” operator (<<) as a friend of the above Polynomial ADT to print out a polynomial on an output stream.For instance, let p = 5.1 \* x^10 + 7.2 \* x^4 - 11.0, the statement "cout<<p;" will print out " 5.1 \* x^10 + 7.2 \* x^4 - 11.0" on the screen.

Lastly, in your main() function, please provide a user-friendly interface that allows the grader to test each and every function implemented above.

**Problem II.  A SquareMatrix ADT.  (50 points)**

A matrix is a 2D array of numerical values. In this problem, you only need to consider square matrices. You can add, subtract or multiply two matrices to form a third matrix provided that the two matrices have the same size. You can also multiply a matrix by a scalar. Design and implement an ADT SquareMatrix using separate compilation to support these operations. Specifically, you are required to include the following member functions in your ADT:

1.     A default constructor that initializes a matrix to 10 by10 with all elements set to 0.0.

2.     A constructor that initializes a matrix by using values stored in a vector of vectors. In other words, this constructor will look like *SquareMatrix(vector< vector<double> > v2d );*  v2d is essentially a 2d matrix.  Your function needs to check whether v2d is square.

3.     A read-only accessor *getValue(int i, int j)*that returns the value at position (i, j) in the matrix.

4.     A mutator *setValue(int i, int j, double value)* that sets the element at position (i, j) to *value*.

5.     An overloaded, read-only multiplication operator (\*) as a member function to multiply two matrices if their dimensionality matches.  Let m1, m2 and m3 be three matrices of the same size.  The following statement “m3= m1 \* m2;” will not change m1 and m2 and will save the multiplication result in m3.

6.     An overloaded, read-only subtraction operator (-) as a member function to subtract one matrix from another if their dimensionality matches. Let m1, m2 and m3 be three matrices of the same size.  The following statement “m3= m1 - m2;” will not change m1 and m2 and will save the subtraction result in m3.

7.    Also include an overloaded “put” operator (<<) as a friend function of the SquareMatrix ADT to print out a matrix on an output stream in a readable format.  For example, let m1 be a 2 by 2 matrix. Its first row contains numbers 11.11 and a1212. Its second row contains numbers 21.21 and 22.22 The statement “cout<<m1;” will print out m1 on the screen in the following format:

                   ( 11.11, 12.12

                     21.21,  22.22)   
Lastly, in your main() function, please provide a user-friendly interface that allows the grader to test each and every function implemented above.