In this assignment we will create two templates, a class template called *Repository* and a function class called *Square*. We will also create two other classes *Matrix* and *Polynomial* to test our templates.

We will create three files, Template.h to contain our class and template definitions, Template.cpp to contain the function definitions for the functions declared in Template.h, and Driver.cpp which will contain the main program.

The template Square will have one *class* argument T. The function will take a parameter x of type T and return x*x, which is also of type T.

The Repository template will have one *class* argument T. The class has three private data items:

1. A pointer of type T named Data.
2. An int named InsertPos
3. An int named Count.

The class also has five public functions. Note that these functions must be function templates and their definitions must be contained in the same file as the class definition.

1. Default constructor
   Assigns an array of size 10 to the pointer Data and the value 10 to Count.
2. Destructor
3. int constructor
   Assigns its argument to Count, and an array of size Count to Data.
4. void Insert(T NewData)
   Adds the new data item to the array Data at index “InsertPos” and
   Then adds 1 to InsertPos. Does nothing if InsertPos >= Count.
   Insert makes a copy of its argument and assigns a pointer to the copy
   To the array.
5. void Print(ostream &OFile)
   Calls “Print” on each object stored in Data.

Class Polynomial
Data items: (private)
- Data: a pointer to a float (to an array of floats)
- Degree: Largest power of x, one smaller than the size of the array.

Functions: (public)
- void ReadData(ifstream &IFile)
  - Read one data item from IFile, which is assigned to Degree.
  - Delete the old array in Data
  - Allocate a new array to Data.
  - Read in enough data items to fill the array, Data.
- void Print(ostream &OFile)
  - First print the line:

" Printing Polynomial of Degree x" where x is the value of Degree.
Print out the items in the array Data, with a space following each one, and
An end of line following the last one.
Print an extra blank line at the end.

Polynomial(int NewDeg)
Create a zero polynomial of degree NewDeg. (Array elements all zero)

Polynomial(const Polynomial &Old)
Standard copy constructor. Allocate new array, don't just copy pointer.

Polynomial operator=(const Polynomial &Old)
Standard assignment overload. Allocate new array don't just copy pointer.

Polynomial operator*(const Polynomial &Old);
Multiply overload. Create new polynomial internally and return it.

Polynomial( )
Create a zero polynomial of degree zero. Data will be an array with one
Element which will be set to zero.

~Polynomial( )
delete Data.

Class Matrix

Data Items (private)

Data: a pointer to a pointer to a float

Dimension: the dimension of the square matrix contained in Data.

Functions (public)

void ReadData(ifstream &IFile)
Delete the old Data.
Read one data item that gives the new value for Dimension.
Allocate a new square matrix
Read sufficient data items to fill the matrix.

void Print(ostream &OFile)
Print the line "Printing a x by x matrix.\n"; where x is the Dimension
Print each row on a separate line with spaces between the numbers
Print an extra blank line at the end.

Matrix(int NewDeg)
Create a zero matrix (all entries zero) of dimension NewDeg.

Matrix(const Matrix &Old)
Standard copy constructor.

Matrix operator=(const Matrix &Old)
Standard assign overload

Matrix operator*(const Matrix &Old)
Multiply overload. Creates a new matrix and returns it.

Matrix( )
Create a zero matrix of size 1x1.

~Matrix( )
deletes the matrix contained in Data.

Main Program.
Open the file "Template.txt". Read in a polynomial (using the ReadData function.) Make a Polynomial Repository of size 5 and insert the polynomial you just read into it. Square the polynomial and add the square to the Repository. Square it again and add the second square to the Repository. Do this same thing twice more.

Read a matrix from "Template.txt". Make a Matrix Repository of size 5, and add the matrix to it. Square the matrix four times and add the results of each squaring to the repository.

Print both repositories.

Notes:

Polynomials are functions of the form $x^2+3x+1$ and $7x^5+3x^4+8x^3+x+1$. The degree of a polynomial is the largest power of x. (2 and 5 in these examples). The coefficients of the polynomial are the numbers preceding $x$, 1, 3, and 1 in the first example, and 7, 3, 8, 0, 1 and 1. The constant term is assumed to be multiplied by $x^0=1$, and missing powers of $x$ are assumed to have coefficients of zero. To hold the coefficients of a degree-n polynomial, we need an array of size n+1. If the array: int P[3]; holds the coefficients of a degree-2 polynomial, P[0] is the constant term, P[1] is the coefficient of $x$, and P[2] is the coefficient of $x^2$. The product of two polynomials can be computed using the following algorithm.

```
Polynomial Polynomial::operator*(const Polynomial &Old)
{
    Polynomial rv(Degree+Old.Degree);
    for (int i = 0 ; i<=Degree ; i++)
    {
            for (int j=0 ; j<=Old.Degree ; j++)
            {
                    rv.Data[j+i] += Data[i]*Old.Data[j];
            }
    }
    return rv;
}
```

To allocate a two-dimensional array, we start with a double pointer:

int **Array;

We will allocate this as an array of arrays. First we allocate an array of pointers and assign that to Array.

Array = new int * [Size];

Next we allocate a secondary array for each row in the two-dimensional array.

```
for (k=0 ; k<Size ; k++)
{
        Array[k] = new int [Size];
        for (m=0 ; m<Size ; m++)
        {
                Array[k][m] = 0;
        }
}
```

To delete a two-dimensional array, we first must delete each row. Then we delete the entire array.

```
for (k=0 ; k<Size ; k++)
{
        delete [] Array[k];
}
delete [] Array;
```

To multiply two square matrices, use the following algorithm.

```
Matrix Matrix::operator*(const Matrix &Old)
{
   if (Dimension != Old.Dimension)
   {
      return Matrix(); // dimensions must match, return 0 matrix.
   }
   else
   {
      Matrix rv(Dimension);
      for (int i=0; i<Dimension ; i++)
      {
         for (int j=0; j<Dimension ; j++)
         {
            rv.Data[i][j] = 0;
            for (int k=0 ; k<Dimension ; k++)
            {
               rv.Data[i][j] += Data[i][k] * Old.Data[k][j];
            }
         }
      }
      return rv;
   }
}
```