C++ Templates -- A Simple Example

Rajanikanth Jammalamadaka<rajani@ece.arizona.edu>

This article describes a C++ code for performing basic operations on matrices using templates. That means we can create a matrix of any data type using one line and call the respective functions.

The code listing *opmatrix.h* is the header file in which the matrix class is described. Note that the number of rows and columns is hard coded in the header file. So, in the current code, a matrix of two rows and two columns has been created. These numbers can be changed for matrices of bigger dimensions. Also for convenience's sake, this code works only for square matrices (matrices which have the same number of rows and columns).

The header file defines the matrix as a two dimensional vector. A vector is a container in C++ which is very similar to an array in the C language but is more sophisticated (it manages its own memory and you can call a number of functions to perform useful operations.) The functions *readm()* and *printm()* are used to read in the elements of the matrix and to print the matrix.

Note that the *readm()* and *printm()* functions use the *this* pointer. The *this* pointer stores the address of the current object. For example, if we declare an object of the matrix class of type int like this

matrix<int> a;

then the *this* pointer will contain the address of the object a, i.e. *this* = &a

Therefore, saying *a.readme()* is equivalent to saying

```
for(int i = 0; i < ROWS; i++)
        for(int j = 0; j < COLS; j++)
                cin >> (&a)->s[i][j];
```

The overloaded operators +,-,* and ~ are declared as friend functions of the matrix, so that they can access the elements of the matrix (which are private) in order to perform the operations. The overloaded operator ~ is the transpose operator.

The + operator operates on two matrices and adds the corresponding elements of the two matrices and prints the result.
The - operator is similar to the + operates except that it subtracts the elements of two matrices, instead of adding them.
The * operator needs some explanation. Matrix multiplication works only if the number of columns of the first matrix is equal to the number of rows of the second matrix. For example, if we have to multiply matrices a and b, then the number of columns of matrix a must be equal to the number of rows of matrix b. In our case this is not a problem

because we are dealing with square matrices(whose sizes are fixed once the variables ROWS and COLUMNS are initialized), so the number of columns of first matrix will always be equal to the number of rows of the second matrix.

In order to understand how the * operator works, let us consider a simple example

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$b = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Now, $a*b = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$

Now, both *a* and *b* are of dimensions 2 by 2. Therefore, their product will be of dimension 2 by 2. The first element of a*b is gotten by summing the product of the corresponding elements of the first row of matrix a and first column of matrix b
i.e.

$$(a*b)_{11} = 1*5 + 2*7 = 5 + 14 = 19.$$

$(a*b)_{ij}$ denotes the element at the intersection of i^{th} row and j^{th} column of the product matrix a*b. Similarly, the $(a*b)_{12}$ is gotten by summing the product of the corresponding elements of the first row of matrix a and second column of matrix b. Similarly the other two elements can be obtained.

The ~ operator transposes the elements of a matrix. By transpose, we mean interchanging the rows and columns of a matrix. So, a matrix $(A)_{ij}$ after the transpose operation becomes $(A)_{ji}$.

For example the matrix $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ after transposing becomes

$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$.

The above examples may seem to be trivial, but they were purposefully made trivial in order to understand the concepts of basic matrix operations. If the matrices were of large dimensions, it wouldn't be a trivial task to multiply them manually. In the results section, operations are performed on two matrices each of dimension 4 by 4. It is here that operator overloading proves to be most useful. For example, if we have to find the transpose of a+b*c, all we need to do is ~(a+b*c) and store this in a matrix and print the resultant matrix using the *printm()* function.

Also note that each operator accepts a constant reference to the matrix, this is because we want each operator to perform its function without modifying the original matrix which was given to it.

Rajanikanth Jammalamadaka

opmatrix.h // C++ header file

```
#include<iostream>
#include<vector>
        using std::cin;
        using std::cout;
        using std::vector;
        using std::endl;

        const int ROWS = 2;
        const int COLS = 2;

        template<class T>
        class matrix
        {
            //declare a vector of vectors of type T
             vector< vector<T> > s ;
          public:

        //Initialize the size of s to ROWS by COLS
                matrix(): s(ROWS, vector<T>(COLS)) {}
                        void readm();
                        void printm();
                //declare the operators +,-,*,~ as  friends and with return type matrix<T>
                        friend matrix<T> operator+<>(const matrix&, const matrix&);
                        friend matrix<T> operator-<>(const matrix&, const matrix&);
                        friend matrix<T> operator*<>(const matrix<T>&, const
matrix<T>&);
                        friend matrix<T> operator~<>(const matrix<T>&);

        };



        template<class T>
        void matrix<T>::readm()
        {
                for(int i = 0; i < ROWS; i++)
                for(int j = 0; j < COLS; j++)
                        cin >> this->s[i][j];
```

```
    }

template<class T>
void matrix<T>::printm()
{
        for(int i = 0; i < ROWS; i++)
         {
                for(int j = 0; j < COLS; j++)
                        cout<< this->s[i][j] <<"\t";
                        cout << endl;
        }
}

        template<class T>
        matrix<T> operator+(const matrix<T>& a, const matrix<T>& b)
        {
                //declare a matrix temp of type T to store the result and return this matrix
                matrix<T> temp;
                for(int i = 0; i < ROWS; i++)
                 for(int j = 0; j < COLS; j++)
                temp.s[i][j] = a.s[i][j] + b.s[i][j];
                        return temp;
        }
        template<class T>
        matrix<T> operator-(const matrix<T>& a, const matrix<T>& b)
        {
                matrix<T> temp;
                for(int i = 0; i < ROWS; i++)
                 for(int j = 0; j < COLS; j++)
                temp.s[i][j] = a.s[i][j] - b.s[i][j];
                return temp;
        }

        template<class T>
        matrix<T> operator*(const matrix<T>& a, const matrix<T>& b)
        {
                matrix<T> temp;
                for(int i = 0; i < ROWS; i++)
                {
                        for(int j = 0; j < COLS; j++)
                        {
                                temp.s[i][j] = 0;
                          for(int k = 0; k < COLS; k++)
                                temp.s[i][j] += a.s[i][k] * b.s[k][j];
                        }
                }
```

```
            return temp;

        }


        template<class T>
         matrix<T> operator~(const matrix<T>& trans)
        {
                matrix<T> temp;
                for(int i = 0; i < ROWS; i++)
                 for(int j = 0; j < COLS; j++)
                temp.s[j][i] = trans.s[i][j];
                return temp;
        }
```

matrix.cpp // C++ Source file

```
#include "opmatrix.hpp"

 int main()
{
   matrix<int> a,b,c;       //we can also declare matrices of type int,float,double etc.

   cout<<"Enter matrix a:"<<endl;
   a.readm();
   cout<<"a is:"<<endl;
   a.printm();
   cout<<"Enter matrix b:"<<endl;
   b.readm();
   cout<<"b is:"<<endl;
   b.printm();

   c = a + b;
   cout<<endl<<"Result of a+b:"<<endl;
   c.printm();
   c = a - b;
     cout<<endl<<"Result of a-b:"<<endl;
    c.printm();
    c = a*b;
  cout<<endl<<"Result of a*b:"<<endl;
    c.printm();
       cout << '\n' << "Result of a+b*c is:"<<'\n';
      (a+b*c).printm();
      c = ~(a+b*c);
```

```
   cout << '\n' << "Result of transpose of a+b*c is:"<<'\n';
  c.printm();
            return 0;
}

//Result
>g++ matrix.cpp
>a.out
Enter matrix a:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
a is:
1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      16
Enter matrix b:
17 18 19 20
21 22 23 24
25 26 27 28
29 30 31 32
b is:
17      18      19      20
21      22      23      24
25      26      27      28
29      30      31      32

Result of a+b:
18      20      22      24
26      28      30      32
34      36      38      40
42      44      46      48

Result of a-b:
-16     -16     -16     -16
-16     -16     -16     -16
-16     -16     -16     -16
-16     -16     -16     -16

Result of a*b:
250     260     270     280
618     644     670     696
986     1028    1070    1112
1354    1412    1470    1528
```

Result of a+b*c is:
61189 63786 66383 68980
74025 77166 80307 83448
86861 90546 94231 97916
99697 103926 108155 112384

Result of transpose of a+b*c is:
61189 74025 86861 99697
63786 77166 90546 103926
66383 80307 94231 108155
68980 83448 97916 112384