Author : Hrushikesh Vasista

e-mail : mailto:hmvasista@gmail.com

3rd Mar, 2014

Contents

INTRODUCTION

The 'matrix' class (henceforth referred to as 'matrix') aims at providing the basic operations involving matrices, which can be used in C++ programs. It makes the programmer easy to realize the mathematical operations as normally done when solving by hand. With 'matrix' its now less tiring to perform calculations on large matrices.

FEATURES

All the features of 'matrix' are provided in the form of 'matrix.h'. So just include 'matrix.h' in your code to access these features:

1. Dimensioning a matrix (by specifying rows and columns at declaration)

The dimensioning of a matrix (order and / or elements) can be done in several ways.

Dimensioning with no order

Syntax: **matrix** <matrix_identifier>;

<matrix_identifier> : Valid C-identifier

This makes <matrix_identifier> a default matrix of order 0x0.

Eg: **matrix** A; //[A] = 0x0

Dimensioning (only order)

Syntax - **matrix** <matrix_identifier>(<row>,<col>);

<matrix_identifier> : Valid C-identifier

<row>,<col> : Integers

This makes <matrix_identifier>, a matrix of order <row> x <col>. All the elements are 0.

Eg: **matrix** A(2,3); //[A] = 2x3

Dimensioning (only order)

Syntax - <matrix_identifier>.**order_ip**(<row>,<col>);

<matrix_identifier> : existing matrix

<row>,<col> : Integers

The function **order_ip()** makes <matrix_identifier>, a matrix of order <row> x <col>, with all elements = 0.

Eg: **float** ele_list[] = {1,2,3,4,5,6};

matrix A(2,3,ele_list); //[A] = 2x3

A.**order_ip**(1,2); //[A] is now 1x2

Dimensioning (only order and elements)

Syntax - **matrix** <matrix_identifier>(<row>,<col>,<elements>);

<matrix_identifier> : Valid C-identifier

<row>,<col> : Integers

<elements> : pointer to float array

This makes <matrix_identifier>, a matrix of order <row> x <col>, with elements = as per the array.

Eg: **float** ele_list[] = {1,2,3,4,5,6};

matrix A(2,3,ele_list); //[A] = 2x3

2. Editing an element in the matrix

Syntax - <matrix_identifier>.**edit**(<new_value>,<row>,<column>);

<matrix_identifier> : existing matrix

<new_value> : float, the new value

<rows>,<columns> : integers, location where to edit. This is 0 indexed.

The **edit**() function replaces the element at the given location with the <new value>.

If the location value is invalid / out of bounds, then no changes would be made in the matrix.

Eg: **matrix** A(3,2); //Makes [A] of order 3x2

[A] = |0 0|

|0 0|

|0 0|

A.**edit**(1.23,2,1); //Makes the element at 3rd row, 2nd column as 1.23

```
[A] =  |0    0|
       |0    0|
       |0  1.23|
```

```
A.edit(98.3,3,2);      //No change, since location is out of bound
A.edit(98.3,-1,2);     //No change, since invalid location
```

3. Accessing an element in the matrix

Syntax - <matrix_identifier>.get_element(<row>,<column>);

<matrix_identifier> : existing matrix

<rows>,<columns> : integers, location where to edit. This is 0 indexed.

The get_element() function returns the element at the location given by with the <row>,<column>.

If the location value is invalid / out of bounds, then an exception of type 'int', with value 0, is thrown.

```
Eg: matrix A(3,2);     //Makes [A] of order 3x2
```

```
[A]    =    |0    0|
            |0    0|
            |0    0|
```

```
A.edit(1.23,2,1);      //Makes the element at 3rd row, 2nd column as 1.23
```

```
[A] =  |0    0|
       |0    0|
       |0  1.23|
```

```
float val;

val = A.get_element(2,1);  //val = 1.23
```

```
try
{
        val = A.get_element(4,5);  //Exception
}
catch(int a)
{
        val=0; //val = 0
}
```

4. Printing the matrix

Syntax - `<matrix_identifier>.display();`

<matrix_identifier> : existing matrix

This prints all the elements of the given matrix of as per order ie <rows> x <columns>.

```
Eg:  matrix A(3,2);       //Makes [A] of order 3x2
     A.display();   //Displays [A]

        0    0
        0    0
        0    0
```

5. Assigning the elements to a matrix

Syntax - `<matrix_identifier>.mat_ele_ip();`

<matrix_identifier> : existing matrix

This accepts all the elements of the given matrix of as per order ie <rows> x <columns>.

```
Eg:  matrix A(3,2);       //Makes [A] of order 3x2
     A.mat_ele_ip();      //Accepts 6 elements for [A]
```

6. Addition of matrices

Syntax - `<matrix_ans> = <matrix_1> + <matrix_2> ;`

<matrix_ans>, <matrix_1>, <matrix_2>: existing matrices

The '+' operation performs the addition of the 2 matrices and returns the resultant matrix, which is assigned to the matrix at LHS. The <matrix_ans> is re-ordered to accommodate the RHS and hence all it's previous data and attributes are lost and filled with the new ones.

Note: If there is a mismatch in the order between <matrix_1> and <matrix_2>, no addition is performed and hence <matrix_ans> remains as before.

Eg: **matrix** A(3,2); //Makes [A] of order 3x2

matrix B(3,2); //Makes [B] of order 3x2

matrix C; //Makes [C] of order 1x1

matrix D(1,4); //Makes [D] of order 1x4

C= A+B; // [C] is now 3x2 and contains A+B.

C= A+D; // [C] remains unchanged, due to mismatch between [A] and [D].

Similarly, '+=' provides addition resultant of one matrix adding itself to another.

Eg: A += B; //same as [A] = [A] + [B]. Contents of [A] are modified

7. Subtraction of matrices

Syntax - <matrix_ans> = <matrix_1> - <matrix_2> ;

<matrix_ans>, <matrix_1>, <matrix_2>: existing matrices

The '-' operation performs the subtraction of the 2 matrices and returns the resultant matrix, which is assigned to the matrix at LHS. The <matrix_ans> is re-ordered to accommodate the RHS and hence all previous data is lost and filled with the new ones.

Note: If there is a mismatch in the order between <matrix_1> and <matrix_2>, no subtraction is performed and hence <matrix_ans> remains as before.

Eg: **matrix** A(3,2); //Makes [A] of order 3x2

matrix B(3,2); //Makes [B] of order 3x2

matrix C; //Makes [C] of order 1x1

matrix D(1,4); //Makes [D] of order 1x4

C= A-B; // [C] is now 3x2 and contains A-B.

C= A-D; // [C] remains unchanged, due to mismatch between [A] and [D].

Similarly, '-=' provides subtraction resultant of 2 matrices.

Eg: A -= B; //same as [A] = [A] - [B]. Contents of [A] are modified

8. Assigning one matrix to another

Syntax - <matrix_1> = <matrix_2> ;

<matrix_1>, <matrix_2>: existing matrices

The '=' operation assigns the matrix on RHS to the matrix on LHS. The <matrix_1> is re-ordered to accommodate the RHS and hence all previous data is lost and filled with the new ones.

Eg: **matrix** A(3,2); //Makes [A] of order 3x2

matrix B; //Makes [B] of order 1x1

A=B; // [A] is now 1x1 and contains elements of [B].

9. Scalar multiplication

Syntax - <matrix_ans> = k*<matrix_1> ;

<matrix_ans> = <matrix_1>*k ;

<matrix_ans>, <matrix_1>: existing matrices

k: float

The '*' operation returns the <matrix_1> with each element multiplied by k .

The <matrix_ans> is re-ordered to accommodate the RHS and hence all previous data is lost and filled with the new ones.

Note: <matrix_1> remains unchanged (both, its order and elements remain unchanged)

Eg: **matrix** A(3,2); //Makes [A] of order 3x2

matrix B; //Makes [B] of order 1x1

B=1.2*A; // [B] is now 3x2 and contains elements of [B] multiplied with 1.2, [A] remains unchanged.

The '*=' operation could be used to perform scalar multiplication of a matrix on itself

Eg: B *= 2; //Same as B = B*2

10. Multiplication of matrices

Syntax - <matrix_ans> = <matrix_1> * <matrix_2> ;

<matrix_ans>, <matrix_1>, <matrix_2>: existing matrices

The '*' operation returns the multiplication of <matrix_1> with <matrix_2>.

The <matrix_ans> is re-ordered to accommodate the RHS and hence all previous data is lost and filled with the new ones.

Note: <matrix_1> and <matrix_2> remain unchanged (both, its order and elements remain unchanged).

If there is an order mismatch, then multiplication is not performed and <matrix_ans> remains unchanged.

Eg: **matrix** A(3,2); //Makes [A] of order 3x2

matrix B(2,3); //Makes [B] of order 2x3

matrix C; //Makes [C] of order 1x1

matrix D; //Makes [D] of order 1x1

C=A*B; // [C] is now 3x3 and contains elements of [A] multiplied with [B].

C=A*D; // [C] is still 3x3 and unchanged. multiplication not performed..

The '*=' operation performs the multiplication and assigns the result to the first operand matrix.

Eg: A *= B; // [A] is now = [A]*[B]

11. Transpose of a matrix

Syntax - <matrix_1>.**transpose**();

<matrix_1>: existing matrices, of order say m x n

This transposes the <matrix_1>.

The <matrix_1> is thus re-ordered to n x m, containing the previous elements itself, but in transposed

manner.

Eg: **matrix** A(3,2); //Makes [A] of order 3x2

A.**transpose**(); // [A] now has an order of 2x3

12. Inverse of a matrix

Syntax - <matrix_2> = <matrix_1>.**inverse**();

<matrix_2>, <matrix_1>: existing matrices

This returns the inverse of the <matrix_1> to <matrix_2>.

The <matrix_2> is thus re-ordered to accommodate the RHS and hence all previous data is lost and filled with the new ones.

Note: The above is true only if <matrix_1> is a square matrix, with its determinant not equal to 0.

In case of <matrix_1> being a non-square matrix / determinant =0, then inverse doesn't exist and hence

<matrix_2> remains unchanged.

Eg: **matrix** A(3,2); //Makes [A] of order 3x2

matrix B(3,3); //Makes [B] of order 3x3

matrix C; //Makes [C] of order 1x1

C= A.**inverse**(); // [C] remains unchanged

C= B.**inverse**(); // [C] is now 3x3 and contains [B]^-1

13. Determinant of a matrix

Syntax - <Det> = <matrix_1>.**determinant**();

<matrix_1> : existing matrix

<det> : float, determinant of <matrix_1>

This returns the determinant of the <matrix_1> to <matrix_2>.

The <det> is contains the determinant of <matrix_1>.

Note: The above is true only if <matrix_1> is a square matrix.In case of <matrix_1> being a non-square matrix, an exception of type 'int' with value 1 is thrown. If the matrix invalid, then exception of type 'int' with value 0 is thrown.

```
Eg:  matrix A(3,2);        //Makes [A] of order 3x2
matrix B(3,3);   //Makes [B] of order 3x3
float det;                 //Makes [C] of order 1x1
det= A.determinant();   // det = 0.
try
{
        det= B.determinant();   //Exception
}
catch (int a)
{
        //Handle  the exception
}
```

14. Covariance of a matrix

Syntax - <matrix_1> = <matrix_2>.**covariance**();

<matrix_1> : Answer matrix

<matrix_2> : Existing matrix containing sample data

This returns the covariance of the <matrix_2> to <matrix_1>.

```
Eg:  matrix X(5,3);        //Makes [X] of order 5X3. User may input the sample data using mat_ele_ip().
```

```
matrix A    ;      //Makes [B]

A = X.covariance();        // [A] contains the covariances of [X]
```

15. Comparison of matrices

Syntax - <matrix_1> == <matrix_2>;

<matrix_1>,<matrix_2> : Existing matrices

The '==' operator could be used to compare 2 matrices and check for their equality.

TRUE is returned if the order and hence the elements of the 2 matrices are equal, else FALSE is returned.

NEW IN v3.1

1. At all places involving dynamic memory allocation (i.e constructors, order_ip(),transpose(), covariance()) exceptions are used to check for failure of allocation. In case of a failure, the execution of the program would cease (i.e. **exit**(1) is called)

CHANGES DONE WRT PREVIOUS VERSION

1. The member function **determinant**() throws an exception if the matrix is 0x0 or non-square matrix (Refer above for details).
2. The member function **get_element**() throws an exception if invalid indices are supplied. (Refer above for details)

HARDWARE / SOFTWARE REQUREMENTS

1. The **'matrix.h'** file can be used on the compilers supporting Standard C++
2. The file **'matrix.h'** must be present in the project folder

KNOWN ISSUES

1. The matrix elements are of type 'float'. Hence the range depends on the type of machine on which the code is running.

   Eg: 16-bit,32-bit, 64-bit may have different range for a 'float'
2. Matrices of type 'char' are not supported, and also the elements cannot be a character.
3. The precision of the float number is compiler dependent.
4. The execution of the program would cease if there is not enough memory.
5. If invalid order for a matrix is entered (eg: 0, -ve values), then a matrix of order 0x0 would be created instead.
6. The default order of a matrix (unless specified with valid order) would be 0x0.
7. If the result of invalid computation is assigned to matrix (LHS), the LHS matrix remains unchanged. If the result produces a valid matrix, then the LHS matrix is assigned with the result and its previous contents and attributes are lost.