Department of Electrical and Computer Engineering
University of Puerto Rico
Mayagüez Campus

ICOM 4035 – Data Structures
Fall 2003

Project #1: Matrix and Vector Algebra
Due Date: 11:59 PM, September 12, 2003

Objectives

1. Understand the design, implementation, usage and limitation of a Matrix ADT and Vector ADT based on fixed-sized two-dimensional arrays.
2. Gain experience implementing abstract data types using already developed data structures.
3. Gain experience with object-oriented programming abstractions, especially constructors and operator overloading

Overview

You will implement and test three classes: **Matrix, Vector** and **Vector3D**. Class Matrix implements an matrix with *n* rows and *m* columns; this is often call a *n x m* matrix. Matrix are often depicted as follows:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}$$

This matrix has size 3 x 4, and each element a_{ij} is a member of the matrix. Notice that the notation a_{ij}, denotes the element at row i, column j. For example, element a_{23} is the element at row 2 and column 3. Likewise, element a_{31} is the element at row 3, column 1. For convenience, we often write a matrix **M** as $M = [a_{ij}]$ to indicate that is a matrix with elements a_{ij}. Notice that the number of rows and columns do not need to be the same, but if they are the matrix is called *square matrix*. You class will keep the current number of row and columns in the matrix. However, each matrix will have a maximum number of rows and columns, defined by the constants Matrix::MAX_ROWS and Matrix::MAX_COLUMNS (these are defined in the file Matrix.h)

Your program will implement the following operations on the Matrix class:

1. Matrix addition

2. Matrix subtraction
3. Matrix multiplication
4. Matrix scalar multiplication
5. Matrix transpose

Each of these operations is defined below.

Matrix addition

Given two matrices $A=[a_{ij}]$ and $B=[b_{ij}]$, then the operation A +B will produce a new matrix $C=[c_{ij}]$, where each value is computed as: $c_{ij}=a_{ij}+b_{ij}$. In this case, both matrices A and B **must** have the same numbers of rows and columns. **Otherwise, you will program will throw an assertion.** The resulting matrix C will have the same number of rows and columns as matrices A and B have.

Matrix subtraction

Given two matrices $A=[a_{ij}]$ and $B=[b_{ij}]$, then the operation A -B will produce a new matrix $C=[c_{ij}]$, where each value is computed as: $c_{ij}=a_{ij}-b_{ij}$. In this case, both matrices A and B **must** have the same numbers of rows and columns. **Otherwise, you will program will throw an assertion.** The resulting matrix C will have the same number of rows and columns as matrices A and B have.

Matrix multiplication

Given a matrix $A=[a_{ij}]$, of size *m x p*, and a matrix $B=[b_{ij}]$, of size *p x n*, then the multiplication operation A * B will produce a new matrix $C=[c_{ij}]$, of size *m x n*, where each value is computed as:

$$c_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+a_{i3}b_{3j}+...+a_{ip}b_{pj}.$$

In this case, the number of columns in matrix A (the value *p*) must be equal to the number of rows in B. **Otherwise, you will program will throw an assertion.** The resulting matrix C will have the same number of rows as matrix A (the value *m*) and the same number of columns as matrix B (the value *n*).

Matrix scalar multiplication

Given a matrix $A=[a_{ij}]$ and a number *r*, then the operation r * A will produce a new matrix $C=[c_{ij}]$, where each value is computed as: $c_{ij}=r*a_{ij}$. The resulting matrix C will have the same number of rows and columns as matrix A.

Matrix transpose

Given a matrix $A=[a_{ij}]$, of *m x n*, then the transpose of *A*, denoted as A^T, is an operation that will produce a new matrix $B=[b_{ij}]$, of size *n x m*, where each value is computed as: $b_{ij}=a_{ji}$. Thus, the number of rows in B is equal to the number of columns in A, and the number of columns in B is equal to the number of rows in A.

Vector class

The Vector class is a class derived from the class Matrix. Vector implement the functionality of n-dimensional vector. An n-dimensional vector can be represented as a n x 1 matrix:

$$\begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \end{bmatrix}$$

In this case, the vector is a 4 x 1 matrix, representing a point in a space with four dimensions. It is often convenient to express a vector V in terms of its dimensions: $V = (a_1, a_2, ..., a_n)$ The Vector class will have method to access and modify each of the components of its dimensions. In addition, the Vector class will implement the following operations:

1. Vector length
2. Vector inner (dot product)
3. Vector angle

The semantics of these operations is defined as follows.

Vector length

Given a vector $V = (a_1, a_2, ..., a_n)$, the length of the vector, denoted by $||V||$, is defined as:

$$\|V\| = \sqrt{a_1^2 + a_2^2 + ... + a_n^2}$$

Vector inner (dot) product

Given vectors $V_1 = (a_1, a_2, ..., a_n)$, and $V_2 = (b_1, b_2, ..., b_n)$, the inner (dot) product between V_1 and V_2 , denoted as $V_1 \bullet V_2$, is defined as:

$$V_1 \bullet V_2 = a_1 b_1 + a_2 b_2 + ... + a_n b_n$$

Vector angle

Given vectors $V_1 = (a_1, a_2, ..., a_n)$, and $V_2 = (b_1, b_2, ..., b_n)$, the angle θ between V_1 and V_2 , is a number for which:

$$\cos(\theta) = \frac{V_1 \bullet V_2}{\| V_1 \| \ \| V_2 \|}$$

The number θ is called the arc cosine, or cosine inverse, and can be computed as follows:

$$\theta = \cos^{-1}\left(\frac{V_1 \bullet V_2}{\| V_1 \| \ \| V_2 \|} \right)$$

C++ provides a function called *acos()* than can be used to compute the arc cosine.

Vector 3D Class

The class Vector3D is derived from the Vector class, and it implements a vector in a three dimensional space. In this case, any instance of the Vector3D will have three dimensions, and hence is a 3 x 1 matrix:

$$V = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \end{bmatrix}$$

For example, the points $P_1 = (1,0,1)$ and $P_2 = (1,0,0)$ are represented, respectively, as:

$$P =_1 \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \qquad P_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

In the Vector3D class, you will implement the cross product operator, and you will have two versions of it. The first version will be a member function that computes the cross product between two vectors v_1 and v_2, and assigns the result to v_1. The second version will be a non-member function that computes the cross product between v_1 and v_2, and then assigns the result to a new vector v_3.

Vector Cross Product

Given vectors $V_1 = (a_1, a_2, a_3)$, and $V_2 = (b_1, b_2, b_3)$, the cross product between V_1 and V_2, denoted as $V_1 \times V_2$, is a vector $V_3 = (c_1, c_2, c_3)$ such that

$$c_1 = a_2 b_3 - a_3 b_2$$
$$c_2 = a_3 b_1 - a_1 b_3$$
$$c_3 = a_1 b_2 - a_2 b_1$$

Distribution Files

You can go to the class web page and download a tar file containing all the files related with this project. Just access the link named Projects, and download the sources files associated with the link: *Project #1– Matrix and Vector Algebra.*

You implementation will consist of adding C++ code to implement three modules: Matrix.cpp, Vector.cpp, and Vector3D.cpp. You will receive all the .h files with declaration of the Matrix, Vector and Vector3D classes. In addition, you will be provided with a main program that uses the Matrix, Vector, and Vector3D classes, and interacts with the user to ask his/her input on the operations to be performed. Finally, you will be given a Makefile with all the commands needed to compile and submit your project.

1. Object.h – declaration of the Object type that will be stored in the matrix or vector.
2. Matrix.h – declaration of the Matrix class.
3. Matrix.cpp – implementation of the Matrix class. YOU MUST IMPLEMENT THE METHODS TO APPEAR IN THIS FILE.

4. Vector.h – declaration of the Vector class.
5. Vector.cpp – implementation of the Vector class. YOU MUST IMPLEMENT THE METHODS TO APPEAR IN THIS FILE.
6. Vector3D.h – declaration of the Vector3D class.
7. Vector3D.cpp – implementation of the Vector3D class. YOU MUST IMPLEMENT THE METHODS TO APPEAR IN THIS FILE.
8. matrix_main.cpp – main program to the Matrix, Vector and Vector3D classes. DO NOT MODIFY THIS FILE!!!!
9. Makefile – file with the commands to compile and submit you project.
10. test1.in – test input file 1.
    NOTE: YOU PROGRAM MUST PASS THIS FILE WITHOUT ERRORS IN ORDER TO BE CONSIDERED A RUNNING PROGRAM.
11. test1.out – expected output from test input file 1.
12. test2.in – test input file 2.
13. test2.out – expected output from test input file 2.
14. test3.in – test input file 3.
15. test3.out – expected output file from test input file 3.
16. prof_matrix_main – professor's version of the matrix_main program. NOTE: Known to be working correctly.

PROJECT DUE DATE: **11:59 PM – September 12, 2003**.