UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS344 - Computer Graphics

Practical 1 Specification: Mathematical Programming

Release Date: 26-02-2024 at 06:00

Due Date: 11-03-2024 at 10:00

Total Marks: 131

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually, no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as **no extension will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or classes).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `http://www.ais.up.ac.za/plagiarism/index.htm`.
- Unless otherwise stated, the usage of additional libraries outside of those indicated in the assignment, will not be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use C++11**.
- All functions should be implemented in the corresponding cpp file. No inline implementation apart from the provided functions.
- Skeleton files have been provided for you with correct linking. Do not change the linking provided, as this may lead to compilation errors.
- Do not change any of the provided functions. This is used by FitchFork to print out your matrices and vectors.
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.

2 Overview

For this practical, you will be implementing matrices and vectors in C++. This will be used to programmatically understand fundamental linear algebra concepts needed for computer graphics.

3 Your Task:

You are required to implement all of the provided functions in the .h file. Section 3.1 explains the object management functions and Section 3.2 explains all the linear algebra functions. Section 3.3 describes the provided functions that will be overwritten.

3.1 Object management functions

This section describes the miscellaneous functions that need to be implemented.

3.1.1 Vector

This class represents a vector[1] in a possible N-D space.

- Vector(int)
  - This constructor should initialize the size of the vector and the array using the passed-in parameter.
  - It is at your discretion with what value the elements in the vector should be initialized to.
  - If the passed-in parameter is non-positive,[2] the function should throw `MathException` `InvalidVectorSize` exception.
- Vector(int, double*)
  - This constructor should initialize the size of the vector and the array using the passed-in parameters.
  - Make use of shallow copies for the array.
  - It can be assumed that the input to this constructor will be valid.
- ~Vector()
  - This destructor should deallocate any dynamically allocated memory.
- Vector(const Vector&)
  - This copy constructor should create a deep copy of the passed-in parameter.
- operator Matrix() const;
  - This is a conversion operator[3] which will return a $N \times 1$ Matrix, populated with the values in the vector.
- int getN() const
  - Getter for the size of the vector.

[1] Please see `https://en.wikipedia.org/wiki/Vector_(mathematics_and_physics)` if you are unsure what a vector is.

[2] Less than or equal to 0

[3] See option 1 given in the table on the page: `https://en.cppreference.com/w/cpp/language/cast_operator` if you can't remember what a conversion operator is.

3.1.2 Matrix

This class represents a row-major[4] 2-D matrix of size $N \times M$.

- Matrix(int, int)
  - This constructor should initialize the `N` member variable with the first passed-in parameter, and the `M` member variable with the second parameter.
  - The constructor should also initialize the array such that it forms a $N \times M$ matrix, populated with values of your choice.
  - If either of the passed-in parameters is non-positive, an `MathException InvalidMatrixSize` exception should be thrown.
- Matrix(int, int, double**)
  - This constructor should use the passed-in parameters to create a $N \times M$ matrix where the first parameter is the `N` value and the second the `M` value.
  - It can be assumed that the parameters will be valid.
  - Use shallow copy to initialize the array member variable.
- Matrix(const Matrix &)
  - This copy constructor should be used to create a deep copy of the passed-in parameter.
- virtual ~Matrix()
  - This virtual destructor should deallocate any dynamically allocated memory.
- int getN() const
  - Getter for the `N` member variable.
- int getM() const
  - Getter for the `M` member variable.

3.1.3 SquareMatrix

This class represents a row-major 2-D matrix of size $N \times N$ and publically inherits from the Matrix class.

- SquareMatrix(int)
  - This is the constructor for the SquareMatrix, where the parameter specifies the size of the matrix.
  - This constructor should initialize all member variables as discussed in the Matrix class, including the throwing of the exception.

[4] See `https://en.wikipedia.org/wiki/Row-_and_column-major_order` for more information.

- SquareMatrix(int, double**)
  - This is the constructor for the SquareMatrix where the parameter specifies the size of the matrix.
  - This constructor should initialize all member variables as discussed in the Matrix class.
  - It can be assumed that all input will be valid.
- virtual ~SquareMatrix()
  - This virtual destructor should deallocate any dynamically allocated memory.
  - *Hint: Revise your COS110 notes with regards to how virtual destructors will be invoked.*

3.1.4 IdentityMatrix

This class represents a row-major 2-D identity matrix[5] of size $N \times N$ and publically inherits from the SquareMatrix class.

- IdentityMatrix(int)
  - This is the constructor for the SquareMatrix, where the parameter specifies the size of the matrix.
  - This constructor should initialize all member variables as discussed in the Matrix class, with the exception that the values in the array should correspond to an identity matrix.
- virtual ~IdentityMatrix()
  - This virtual destructor should deallocate any dynamically allocated memory.
  - *Hint: Revise your COS110 notes with regards to how virtual destructors will be invoked.*

3.2 Linear algebra functions

This section describes the linear algebra functions you will be implementing. Assume the following matrices, vectors, and scalars[6] have been defined:
Matrices: $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{n \times r}$
Vectors: $v, w, t \in \mathbb{R}^n$
Scalars: $s \in \mathbb{R}$
Your task will be to implement the functions as described in Table 1.

[5]Please see `https://en.wikipedia.org/wiki/Identity_matrix` if you are unsure what an identity matrix is.
[6]Please see `https://en.wikipedia.org/wiki/Scalar_(mathematics)` if you are unsure what a scalar is.

Mathematical notation	Function	Description
$v + w$	Vector::operator+	Returns a new vector, which is the result of the vector addition performed on the `this` vector (v) and the passed-in vector (w). See point 7.
$v - w$	Vector::operator-	Returns a new vector, which is the result of the vector subtraction performed the `this` vector (v) and the passed-in vector (w). See point 7.
$v \times s$	Vector::operator*	Returns a new vector which is the result of vector scaling between the `this` vector (v) and the passed-in scalar (s).
$v \cdot w$	Vector::operator*	Returns the dot product between the `this` vector (v) and the passed-in vector (w). See point 6.
$\|\|v\|\|$	Vector::magnitude	Returns the magnitude of the `this` vector (v). See point 8.
$v \times w$	Vector::crossProduct	Returns the cross-product between the `this` vector (v) and the passed-in vector (w). See point 5.
$\hat{v}$	Vector::unitVector	Returns the unit vector of the `this` vector (v).[4]
$A \times B$	Matrix::operator*	Returns a new matrix, that is the result of the matrix multiplication performed on the `this` matrix (A) and the passed-in matrix (B). See point 3.
$A \times s$	Matrix::operator*	Returns a new matrix, that is the result of scaling the `this` matrix (A) with the passed-in scalar (s).
$A + B$	Matrix::operator+	Returns a new matrix, that is the result of the element wise summation of the `this` matrix (A) with the passed-in matrix (B). See point 2.
A^T	Matrix::operator~	Returns a new matrix, that is the transpose of the `this` matrix (A).
A^{-1}	SquareMatrix::operator!	Returns a new matrix, that is the inverse of the `this` matrix (A).
$An = t$	SquareMatrix::solve	This function should solve the system of linear equations, where the `this` matrix (A) is the coefficient matrix, the passed-in vector (t) is the constants. The function should return the values for the unknown variables in vector form (n).
$det(A)$	SquareMatrix::determinant	This function should return the determinant of the `this` matrix (A)

Table 1: Table explaining all the linear algebra functions that need to be implemented

Take note of the following:

1. Any matrix, that is part of the SquareMatrix class, is a $n \times n$ matrix.
2. For the matrix addition operator, if the passed-in matrix is not the same shape as the current matrix, throw the `MathException InvalidMatrixAddition` exception.
3. For the matrix multiplication operator, if the sizes of the matrices do not allow matrix multiplication, throw the `MathException InvalidMatrixMultiplication` exception.
4. For the unit vector function, if the magnitude of the vector is 0, then throw the `MathExceptions InvalidUnitVector` exception.
5. For the cross product function, if either of the vectors involved in the function does not have a size of 3, throw the `MathException InvalidCrossProduct` exception.
6. For the vector dot-product function, if the two vectors are not of the same size, throw the `MathExceptions InvalidDotProduct` exception.
7. For the vector addition and vector subtraction function, if the two vectors are not of the same size, throw the `MathExceptions InvalidVectorAddition` exception.
8. For the magnitude of the vector, use the Euclidean norm[7] for your calculations.
9. If the determinant of the matrix if 0, then throw the `MathExceptions UnsolvableSystemOfLinearEquations` exception.

3.3 Provided functions

You are provided with the following functions in the .h files. **Do not** change these functions as they will be overwritten on FitchFork.

3.3.1 Vector

- Subscript operator
  - This can be used to access and set elements in the vector.
- Print function
  - This function is used by the marking script to print out the vector.

3.3.2 Matrix

- Subscript operator
  - This can be used to access and set elements in the vector.
- Print function

[7] Please see `https://en.wikipedia.org/wiki/Norm_(mathematics)` for more information.

  - This function is used by the marking script to print out the vector.

You are also provided with the MathException class.

4 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **C++11**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- Do not include using namespace std in any of the files.
- You may only use the following libraries:
  - IOStream
  - CMath
- You will notice that each submission uses randomly generated matrices and vectors. This is to allow you to view your output on FitchFork and see where your code failed.
- Ensure you do not have any debugging output as this can interfere with the marking script.
- The exceptions will not be explicitly tested. They are there to assist you while testing your own code such that you are able to find errors easier.
- When adding helper functions, add them as global functions in the .cpp files. This implies you will need to use the public interface to access member variables of objects. It is highly recommended that you make use of global helper functions.

5 Upload Checklist

The following C++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- Matrix.h
- Matrix.cpp
- Vector.h
- Vector.cpp
- MathException.h

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

Skeleton files have been provided for you with correct linking. Do not change the linking provided, as this may lead to compilation errors. Also, do not change any of the provided functions. This is used by FitchFork to print out your matrices and vectors.

6 Submission

You need to submit your source files on the FitchFork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ -std=c++11 -g *.cpp -o main
```

and run with the following command:

```
./main
```

Remember your .h file will be overwritten, so ensure you do not alter the provided .h files.
You have 20 submissions and your best mark will be your final mark. Upload your archive to the Practical 1 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**