Swinburne University of Technology

School of Science, Computing and Engineering Technologies

ASSIGNMENT COVER SHEET

Subject Code:	COS30008
Subject Title:	Data Structures and Patterns
Assignment number and title:	1, Solution Design in C++
Due date:	Wednesday, March 27, 2024, 23:59
Lecturer:	Dr. Markus Lumpe

Your name: Nur E Siam **Your student ID:** 103842784

Marker's comments:

Problem	Marks	Obtained
1	26	
2	98	
3	32	
Total	156	

Extension certification:

This assignment has been given an extension and is now due on ____________

Signature of Convener:________________

```cpp
// Vector3D_PS1.cpp
// Created by NUR E SIAM

#include "Vector3D.h"
#include <sstream>

std::string Vector3D::toString() const noexcept {
    // Create a stringstream to build the output string
    std::stringstream stream;

    // Format x, y, and w components with 4 decimal places and add to the stream
    stream << "[" << std::round(x() * 10000.0f) / 10000.0f << ","
        << std::round(y() * 10000.0f) / 10000.0f << ","
        << std::round(w() * 10000.0f) / 10000.0f << "]";

    // Extract the formatted string from the stringstream
    std::string output = stream.str();

    // Return the formatted string representation of the vector
    return output;
}

```

```cpp
// Matrix3x3_PS1.cpp
// Created By NUR E SIAM

#include "Matrix3x3.h"
#include <iostream>

// Matrix multiplication implementation
Matrix3x3 Matrix3x3::operator*(const Matrix3x3& aOther) const noexcept {
    // Calculate the product of two matrices without using loops
    // Each entry in the resulting matrix is the dot product of a row from the first matrix
    // and a column from the second matrix

    // Calculate the dot products for each entry of the resulting matrix
    float result00 = row(0).dot(aOther.column(0));
    float result01 = row(0).dot(aOther.column(1));
    float result02 = row(0).dot(aOther.column(2));

    float result10 = row(1).dot(aOther.column(0));
    float result11 = row(1).dot(aOther.column(1));
    float result12 = row(1).dot(aOther.column(2));

    float result20 = row(2).dot(aOther.column(0));
    float result21 = row(2).dot(aOther.column(1));
    float result22 = row(2).dot(aOther.column(2));

    // Create a new matrix with the calculated entries
    Matrix3x3 result(Vector3D(result00, result01, result02),
        Vector3D(result10, result11, result12),
        Vector3D(result20, result21, result22));

    return result;
}

// Determinant calculation implementation
float Matrix3x3::det() const noexcept {
    float a = fRows[0][0], b = fRows[0][1], c = fRows[0][2];
    float d = fRows[1][0], e = fRows[1][1], f = fRows[1][2];
    float g = fRows[2][0], h = fRows[2][1], i = fRows[2][2];
    return a * (e * i - f * h) - b * (d * i - f * g) + c * (d * h - e * g);
}

// Check if the matrix has an inverse
bool Matrix3x3::hasInverse() const noexcept {
    return det() != 0.0f;
}

Matrix3x3 Matrix3x3::transpose() const noexcept {
    Matrix3x3 result(Vector3D(fRows[0][0], fRows[1][0], fRows[2][0]),
        Vector3D(fRows[0][1], fRows[1][1], fRows[2][1]),
        Vector3D(fRows[0][2], fRows[1][2], fRows[2][2]));
    return result;
}

Matrix3x3 Matrix3x3::inverse() const noexcept {
    if (det() == 0) {
        throw std::runtime_error("Inverse does not exist.");
    }
    float result00 = fRows[1][1] * fRows[2][2] - fRows[1][2] * fRows[2][1];
    float result01 = fRows[0][2] * fRows[2][1] - fRows[0][1] * fRows[2][2];
    float result02 = fRows[0][1] * fRows[1][2] - fRows[0][2] * fRows[1][1];

    float result10 = fRows[1][2] * fRows[2][0] - fRows[1][0] * fRows[2][2];
    float result11 = fRows[0][0] * fRows[2][2] - fRows[0][2] * fRows[2][0];
    float result12 = fRows[0][2] * fRows[1][0] - fRows[0][0] * fRows[1][2];

    float result20 = fRows[1][0] * fRows[2][1] - fRows[1][1] * fRows[2][0];
    float result21 = fRows[0][1] * fRows[2][1] - fRows[0][0] * fRows[2][1];
    float result22 = fRows[0][0] * fRows[1][1] - fRows[0][1] * fRows[1][0];

    Matrix3x3 mult(Vector3D(result00, result01, result02),
        Vector3D(result10, result11, result12),
        Vector3D(result20, result21, result22));

    Matrix3x3 output = mult * (1 / det());
    return output;

}

std::ostream& operator<<(std::ostream& os, const Matrix3x3& matrix) {
    // Output each row of the matrix
    os << "[";
    for (int i = 0; i < 3; ++i) {
        os << matrix.fRows[i].toString();
        if (i < 2) {
            os << ",";
        }
    }
    os << "]";
    return os;
}
```

```cpp
//Polygon_PS1.cpp

// Created by NUR E SIAM

#include "Polygon.h"
#include <cmath>


float Polygon::getSignedArea() const noexcept {
    // Calculates the signed area of the polygon using the shoelace formula
    float larea = 0.0f;

    for (size_t lIndex = 0; lIndex < fNumberOfVertices; ++lIndex) {
        size_t j = (lIndex == fNumberOfVertices - 1) ? 0 : lIndex + 1; // Wrap around for closing edge

        larea += fVertices[lIndex].x() * fVertices[j].y() -
            fVertices[j].x() * fVertices[lIndex].y();
    }

    return larea / 2.0f;
}

Polygon Polygon::transform(const Matrix3x3& aMatrix) const noexcept {
    // Creates a new polygon by applying the given transformation matrix to each vertex
    Polygon ltransformed;
    ltransformed.fNumberOfVertices = fNumberOfVertices;

    size_t lIndex = 0;
    size_t rIndex = fNumberOfVertices - 1; // Start from opposite ends

    while (lIndex <= rIndex) {
        // Transform vertices from left to right
        Vector3D vertex = aMatrix * Vector3D(fVertices[lIndex].x(), fVertices[lIndex].y(), 1.0f);
        ltransformed.fVertices[lIndex] = Vector2D(vertex.x(), vertex.y());

        // Transform vertices from right to left (for potential efficiency)
        vertex = aMatrix * Vector3D(fVertices[rIndex].x(), fVertices[rIndex].y(), 1.0f);
        ltransformed.fVertices[rIndex] = Vector2D(vertex.x(), vertex.y());

        lIndex++;
        rIndex--;
    }

    return ltransformed;
}
```