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: Ruffin Remad **Your student ID:** 103840173

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

std::string Vector3D::toString() const noexcept {
    std::stringstream lString;
    lString << "[" << std::round(x() * 10000.0f) / 10000.0f << ","
        << std::round(y() * 10000.0f) / 10000.0f << ","
        << std::round(w() * 10000.0f) / 10000.0f << "]";
    return lString.str();
}

```

```cpp
#include "Matrix3x3.h"
#include <cmath>
#include <cassert>

Matrix3x3 Matrix3x3::operator*(const Matrix3x3& aOther) const noexcept {
    return Matrix3x3(
        Vector3D(fRows[0].dot(aOther.column(0)), fRows[0].dot
          (aOther.column(1)), fRows[0].dot(aOther.column(2))),
        Vector3D(fRows[1].dot(aOther.column(0)), fRows[1].dot
          (aOther.column(1)), fRows[1].dot(aOther.column(2))),
        Vector3D(fRows[2].dot(aOther.column(0)), fRows[2].dot
          (aOther.column(1)), fRows[2].dot(aOther.column(2)))
    );

}

float Matrix3x3::det() const noexcept {
    return fRows[0][0] * (fRows[1][1] * fRows[2][2] - fRows[1][2] *
      fRows[2][1]) -
        fRows[0][1] * (fRows[1][0] * fRows[2][2] - fRows[1][2] * fRows
          [2][0]) +
        fRows[0][2] * (fRows[1][0] * fRows[2][1] - fRows[1][1] * fRows
          [2][0]);
}

Matrix3x3 Matrix3x3::transpose() const noexcept{
    return Matrix3x3(
        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])
    );
}

bool Matrix3x3::hasInverse() const noexcept {
    return det() != 0.0f;
}

Matrix3x3 Matrix3x3::inverse() const noexcept {
    // Compute the determinant of the matrix
    float lDetValue = det();

    // Check if the determinant is zero
    assert(lDetValue != 0.0f);

    // Calculate the inverse matrix using the determined determinant
    float lInvDet = 1.0f / lDetValue;

    return Matrix3x3(
        Vector3D((fRows[1][1] * fRows[2][2] - fRows[1][2] * fRows[2][1])
           * lInvDet,
            (fRows[0][2] * fRows[2][1] - fRows[0][1] * fRows[2][2]) *
              lInvDet,
            (fRows[0][1] * fRows[1][2] - fRows[0][2] * fRows[1][1]) *
```

```cpp
                lInvDet),

        Vector3D((fRows[1][2] * fRows[2][0] - fRows[1][0] * fRows[2][2]) * lInvDet,
            (fRows[0][0] * fRows[2][2] - fRows[0][2] * fRows[2][0]) * lInvDet,
            (fRows[0][2] * fRows[1][0] - fRows[0][0] * fRows[1][2]) * lInvDet),

        Vector3D((fRows[1][0] * fRows[2][1] - fRows[1][1] * fRows[2][0]) * lInvDet,
            (fRows[0][1] * fRows[2][0] - fRows[0][0] * fRows[2][1]) * lInvDet,
            (fRows[0][0] * fRows[1][1] - fRows[0][1] * fRows[1][0]) * lInvDet)
    );
}

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

```

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

float Polygon::getSignedArea() const noexcept {
    float lArea = 0.0f;
    for (size_t i = 0; i < fNumberOfVertices; ++i) {
        size_t j = (i + 1) % fNumberOfVertices;
        lArea += fVertices[i].x() * fVertices[j].y();
        lArea -= fVertices[j].x() * fVertices[i].y();
    }
    return lArea / 2.0f;
}

Polygon Polygon::transform(const Matrix3x3& aMatrix) const noexcept {
    Polygon lTransformedPolygon;
    for (size_t i = 0; i < fNumberOfVertices; ++i) {
        const Vector3D& lVertex3D = Vector3D(fVertices[i].x(), fVertices[i].y(), 1.0f);
        Vector3D lTransformedVertex = aMatrix * lVertex3D;
        lTransformedPolygon.fVertices[i] = Vector2D(lTransformedVertex.x(), lTransformedVertex.y());
    }
    lTransformedPolygon.fNumberOfVertices = fNumberOfVertices;
    return lTransformedPolygon;
}
```