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: Dang Khoa Le **Your student ID:** 103844421

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:_____

```
#define _USE_MATH_DEFINES     // must be defined before any #include
#include "Vector3D.h"
#include <sstream>
#include <iomanip>
#include <cmath>

std::string Vector3D::toString() const noexcept {
    std::stringstream ss;

    // Check if it is an integer
    bool isInteger = (std::floor(x()) == x()) && (std::floor(y()) == y()) && (std::floor(w())
== w());

       // Vector a as and integer has 0 decimal places
       if (isInteger) {
           ss << std::setprecision(0);
       // Non-integer values return as 4 decimal places.
       } else {
           ss << std::fixed << std::setprecision(4);
       }

       // Format output for x and y (as default):
       ss << "[" << x() << "," << y() << ",";

       // Format output for w (by which's whole number has more than 4 digits)
       if (std::abs(w()) > 9999) {
           ss << std::setprecision(1) << w(); // If w()'s whole number has 5 digits, set
precision to 1
       } else {
           ss << w(); // Otherwise, set at default
       }
       ss << "]";

    // Return the resulting string
    return ss.str();
}
```

```cpp
#define _USE_MATH_DEFINES     // must be defined before any #include
#include "Matrix3x3.h"
#include <cmath>
#include <sstream>
#include <iomanip>

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

float Matrix3x3::det() const noexcept
{
    const Vector3D& aRow1 = row(0);
    const Vector3D& aRow2 = row(1);
    const Vector3D& aRow3 = row(2);

    return aRow1[0] * (aRow2[1] * aRow3[2] - aRow2[2] * aRow3[1]) -
           aRow1[1] * (aRow2[0] * aRow3[2] - aRow2[2] * aRow3[0]) +
           aRow1[2] * (aRow2[0] * aRow3[1] - aRow2[1] * aRow3[0]);
}

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

bool Matrix3x3::hasInverse() const noexcept
{
    // Matrix is inversible if the determinant is non-zero
    return det() != 0.0f;
}

Matrix3x3 Matrix3x3::inverse() const noexcept
{
    // Calculate determinant
    float determinant = det();

    // Throw error message back when the matrix is not invertable in tester
    if (determinant == 0.0f)
        throw std::logic_error("Matrix is not invertible");

    // Calculate inverse
    float invDet = 1.0f / determinant;

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

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

// This part was modified from Vector3D.cpp as it returns which a 3x3 vector instead of 3x2
vector.
std::ostream& operator<<(std::ostream& aOStream, const Matrix3x3& aMatrix) {
    aOStream << "[" << aMatrix.row(0) << "," << aMatrix.row(1) << "," << aMatrix.row(2) <<
"]";
    return aOStream;
}

// Non-integer values return with 4 decimal places.
std::ostream& operator<<(std::ostream& aOStream, const Vector3D& aVector) {
    if (std::floor(aVector.x()) == aVector.x() &&
        std::floor(aVector.y()) == aVector.y() &&
        std::floor(aVector.w()) == aVector.w()) {
        aOStream << "(" << aVector.x() << ","
                         << aVector.y() << ","
                         << aVector.w() << ")";
    } else {
        aOStream << "(" << std::fixed << std::setprecision(4) << aVector.x() << ","
                         << std::fixed << std::setprecision(4) << aVector.y() << ","
                         << std::fixed << std::setprecision(4) << aVector.w() << ")";
    }
    return aOStream;
}
```

```cpp
#define _USE_MATH_DEFINES     // must be defined before any #include
#include "Polygon.h"
#include "Matrix3x3.h"
#include <cmath>

float Polygon::getSignedArea() const noexcept {
    float area = 0.0f;
    for (size_t i = 0; i < fNumberOfVertices; ++i) {
        const Vector2D& current = fVertices[i];
        const Vector2D& next = fVertices[(i + 1) % fNumberOfVertices];
        area += (current.y() + next.y()) * (current.x() - next.x());
    }
    return area / 2.0f;
}

Polygon Polygon::transform(const Matrix3x3& aMatrix) const noexcept {
    Polygon transformedPolygon;
    for (size_t i = 0; i < fNumberOfVertices; ++i) {
        const Vector3D vertex3D(fVertices[i].x(), fVertices[i].y(), 1.0f);
        const Vector3D transformedVertex = aMatrix * vertex3D;
        transformedPolygon.fVertices[i] = Vector2D(transformedVertex.x(),
transformedVertex.y());
    }
    transformedPolygon.fNumberOfVertices = fNumberOfVertices;
    return transformedPolygon;
}
```