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: Xuan Tuan Minh Nguyen **Your student ID:** 103819212

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
#pragma once
#include "Vector3D.h"
#include <sstream>
#include <math.h>



std::string Vector3D::toString() const noexcept {
    std::stringstream ss;
    auto roundByTenSqrtFour = [](float roundNum) {
        float sqrtFour = 10000;
        float rounded = round(roundNum * sqrtFour) / sqrtFour;

        return rounded;
    };
    ss << "[" << roundByTenSqrtFour(x()) << ", " << roundByTenSqrtFour(y())
       << ", " << roundByTenSqrtFour(w()) << "]";
    return ss.str();

}



```

```cpp

#include "Polygon.h"

float Polygon::getSignedArea() const noexcept {

    float result = 0.0f;
    if (fNumberOfVertices > 2) {
        for (size_t i = 0; i < fNumberOfVertices; i++)
        {
            size_t j = (i + 1) % fNumberOfVertices;
            result += 0.5 * ((fVertices[i].x() * fVertices[j].y()) -
              (fVertices[i].y() * fVertices[j].x()));
        }
    }

    return result;
}

Polygon Polygon::transform(const Matrix3x3& aMatrix) const noexcept {
    Polygon transformedPolygon = *this;

    for (size_t i = 0; i < fNumberOfVertices; ++i) {

        Vector3D transformedVertex = aMatrix * Vector3D(fVertices[i].x(),
          fVertices[i].y(), 1.0f);
        transformedPolygon.fVertices[i] = Vector2D(transformedVertex.x(),
          transformedVertex.y());
    }

    transformedPolygon.fNumberOfVertices = fNumberOfVertices;

    return transformedPolygon;

}


```

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

Matrix3x3 Matrix3x3::operator*(const Matrix3x3& aOther) const noexcept {

    Matrix3x3 result;
    for (size_t i = 0; i < 3; i++) {
        float* vectorComponents = new float[3];
        for (size_t j = 0; j < 3; j++) {
            vectorComponents[j] = row(i).dot(aOther.column(j));
        }
        result.fRows[i] = Vector3D(vectorComponents[0], vectorComponents[1], vectorComponents[2]);
        delete[] vectorComponents;
    }

    return result;

}

float Matrix3x3::det() const noexcept {
    float m11 = row(0)[0];
    float m12 = row(0)[1];
    float m13 = row(0)[2];
    float m21 = row(1)[0];
    float m22 = row(1)[1];
    float m23 = row(1)[2];
    float m31 = row(2)[0];
    float m32 = row(2)[1];
    float m33 = row(2)[2];

    return (m11 * ((m22 * m33) - (m23 * m32))) - (m12 * ((m21 * m33) - (m23 * m31))) + (m13 * ((m21 * m32) - (m22 * m31)));
}

Matrix3x3 Matrix3x3::transpose() const noexcept {
    return Matrix3x3(
        column(0),
        column(1),
        column(2)
    );

}

bool Matrix3x3::hasInverse() const noexcept {
    return det() != 0;
}

Matrix3x3 Matrix3x3::inverse() const noexcept {
```

```
48      assert(hasInverse());
49      float inverseOfDet = 1.0f / det();
50      float m11 = row(0)[0];
51      float m12 = row(0)[1];
52      float m13 = row(0)[2];
53      float m21 = row(1)[0];
54      float m22 = row(1)[1];
55      float m23 = row(1)[2];
56      float m31 = row(2)[0];
57      float m32 = row(2)[1];
58      float m33 = row(2)[2];
59
60      return Matrix3x3(
61          Vector3D((m22 * m33) - (m23 * m32), (m13 * m32) - (m12 * m33), (m12 ↵
                * m23) - (m13 * m22)),
62          Vector3D((m23 * m31) - (m21 * m33), (m11 * m33) - (m13 * m31), (m13 ↵
                * m21) - (m11 * m23)),
63          Vector3D((m21 * m32) - (m22 * m31), (m12 * m31) - (m11 * m32), (m11 ↵
                * m22) - (m12 * m21))
64      ) * inverseOfDet;
65  }
66
67  std::ostream& operator<<(std::ostream& aOStream, const Matrix3x3& aMatrix) ↵
      {
68      aOStream << "[";
69      for (size_t i = 0; i < 2; i++) {
70          aOStream << aMatrix.row(i).toString() << ", ";
71      }
72      aOStream << aMatrix.row(2).toString();
73      aOStream << "]";
74      return aOStream;
75  }
```