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: H M Asfaq Ahmed Shihab **Your student ID:** 103849248

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
//  problem_Sets
//  Created by H M Asfaq Ahmed Shihab on 23/3/2024.

#define _USE_MATH_DEFINES
#include "Vector3D.hpp"
#include <sstream>
#include <iomanip>

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

    if (std::floor(x()) == x()) {
        ss << std::fixed << std::setprecision(0) << x();
    } else {
        ss << std::fixed << std::setprecision(4) << x();
    }
    ss << ", ";

    if (std::floor(y()) == y()) {
        ss << std::fixed << std::setprecision(0) << y();
    } else {
        ss << std::fixed << std::setprecision(4) << y();
    }
    ss << ", ";

    if (std::floor(w()) == w()) {
        ss << std::fixed << std::setprecision(0) << w();
    } else {
        ss << std::fixed << std::setprecision(4) << w();
    }


    ss << "]";
    return ss.str();
}
```

```cpp
//  Matrix3x3_PS1.cpp
//  problem_Sets
//  Created by H M Asfaq Ahmed Shihab on 23/3/2024.
#define _USE_MATH_DEFINES
#include "Matrix3x3.hpp"
#include <cassert>
#include <cmath>
#include <iostream>

using namespace std;

// Multiplication of matrix
Matrix3x3 Matrix3x3::operator*(const Matrix3x3& aOther) const noexcept {
    Vector3D lRow1(row(0).dot(aOther.column(0)),
     row(0).dot(aOther.column(1)), row(0).dot(aOther.column(2)));
    Vector3D lRow2(row(1).dot(aOther.column(0)),
     row(1).dot(aOther.column(1)), row(1).dot(aOther.column(2)));
    Vector3D lRow3(row(2).dot(aOther.column(0)),
     row(2).dot(aOther.column(1)), row(2).dot(aOther.column(2)));
    return Matrix3x3(lRow1, lRow2, lRow3);
}

// Determinant of the matrix
float Matrix3x3::det() const noexcept {
    const Vector3D& lRow1 = row(0);
    const Vector3D& lRow2 = row(1);
    const Vector3D& lRow3 = row(2);
    return lRow1.x() * (lRow2.y() * lRow3.w() - lRow2.w() * lRow3.y()) -
        lRow1.y() * (lRow2.x() * lRow3.w() - lRow2.w() * lRow3.x()) +
        lRow1.w() * (lRow2.x() * lRow3.y() - lRow2.y() * lRow3.x());
}

// Transpose of the matrix
Matrix3x3 Matrix3x3::transpose() const noexcept {
    return Matrix3x3(column(0), column(1), column(2));
}

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

// Inverse matrix
Matrix3x3 Matrix3x3::inverse() const noexcept {
    assert(hasInverse());
    float detInv = 1 / det();
    Vector3D lInvRow1(
        (row(1).y() * row(2).w() - row(1).w() * row(2).y()) * detInv,
        (row(0).w() * row(2).y() - row(0).y() * row(2).w()) * detInv,
        (row(0).y() * row(1).w() - row(0).w() * row(1).y()) * detInv);
    Vector3D lInvRow2(
        (row(1).w() * row(2).x() - row(1).x() * row(2).w()) * detInv,
        (row(0).x() * row(2).w() - row(0).w() * row(2).x()) * detInv,
        (row(0).w() * row(1).x() - row(0).x() * row(1).w()) * detInv);
    Vector3D lInvRow3(
```

```
        (row(1).x() * row(2).y() - row(1).y() * row(2).x()) * detInv,
        (row(0).y() * row(2).x() - row(0).x() * row(2).y()) * detInv,
        (row(0).x() * row(1).y() - row(0).y() * row(1).x()) * detInv);
    return Matrix3x3(lInvRow1, lInvRow2, lInvRow3);
}

// Output
std::ostream& operator<<(std::ostream& os, const Matrix3x3& matrix) {
    os << "Matrix 3x3:" << endl;
    os << matrix.row(0).toString() << endl
        << matrix.row(1).toString() << endl
        << matrix.row(2).toString() << endl;
    return os;
}
```

```cpp
//  Ploygon_PS1.cpp
//  problem_Sets
//  Created by H M Asfaq Ahmed Shihab on 24/3/2024.

#define _USE_MATH_DEFINES
#include <cassert>
#include <iostream>
#include "Polygon.hpp"
#include "Vector3D.hpp"

float Polygon::getSignedArea() const noexcept {
    if (fNumberOfVertices <= 2) {
        std::cerr << "Error: Polygon must have at least three vertices to
         calculate signed area.\n";
        return 0.0f;
    }

    float lArea = 0.0f;
    for (size_t i = 0; i < fNumberOfVertices - 1; i++) {
        lArea += 0.5f * (fVertices[i].y() + fVertices[i + 1].y()) *
         (fVertices[i].x() - fVertices[i + 1].x());
    }
    lArea += 0.5f * (fVertices[fNumberOfVertices - 1].y() +
     fVertices[0].y()) * (fVertices[fNumberOfVertices - 1].x() -
     fVertices[0].x());
    return lArea;
}

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