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:	Monday, March 27, 2023, 10:30
Lecturer:	Dr. Markus Lumpe

Your name: Ekrar Efaz **Your student ID:** 103494172

Check Tutorial	Tues 08:30	Tues 10:30	Tues 12:30 BA603	Tues 12:30 ATC627	Tues 14:30	Wed 08:30	Wed 10:30	Wed 12:30	Wed 14:30	Thurs 08:30	Thurs 10:30
	□	□	□	□	□	□	□	□	☑	□	□

Marker's comments:

Problem	Marks	Obtained
1	84	
2	32	
Total	116	

Extension certification:

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

Signature of Convener:____________________

```
//
//  Matrix3x3_PS1.cpp
//  Assignment1
//
//  Created by Ekrar Efaz on 20/3/23.
//

#define _USE_MATH_DEFINES     // must be defined before any #include

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

Matrix3x3 Matrix3x3::operator*( const Matrix3x3& aOther ) const noexcept{ // Multiplication
    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{  // Determinant

    // Without Loop

    float a11 = fRows[0][0];
    float a12 = fRows[0][1];
    float a13 = fRows[0][2];

    float a21 = fRows[1][0];
    float a22 = fRows[1][1];
    float a23 = fRows[1][2];

    float a31 = fRows[2][0];
    float a32 = fRows[2][1];
    float a33 = fRows[2][2];

    float det = a11 * (a22 * a33 - a32 * a23)
              - a12 * (a21 * a33 - a31 * a23)
              + a13 * (a21 * a32 - a31 * a22);

    return det;


// With For Loop (working but not efficient)
//    int aMatArray[3][3];
//
//    for(int row=0;row<3;++row){
//        for(int col=0;col<3;++col){
//            aMatArray[row][col] = fRows[row][col];
//        }
//    }
//
//    float determinant = (aMatArray[0][0] * (aMatArray[1][1]*aMatArray[2][2] - aMatArray[2][1]*aMatArray[1][2])
//                      - aMatArray[0][1] * (aMatArray[1][0]*aMatArray[2][2] - aMatArray[2][0]*aMatArray[1][2])
//                         + aMatArray[0][2] * (aMatArray[1][0]*aMatArray[2][1] - aMatArray[2][0]*aMatArray[1][1]));
//
//    return determinant;
}

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

Matrix3x3 Matrix3x3::transpose() const noexcept{
    Vector3D aRow1 = column(0);
    Vector3D aRow2 = column(1);
    Vector3D aRow3 = column(2);

    return Matrix3x3(aRow1,aRow2,aRow3);
}

Matrix3x3 Matrix3x3::inverse() const{
    assert(hasInverse());

    Matrix3x3 cofactor(
                       Vector3D(fRows[1][1] * fRows[2][2] - fRows[1][2] * fRows[2][1],
                                fRows[1][2] * fRows[2][0] - fRows[1][0] * fRows[2][2],
                                fRows[1][0] * fRows[2][1] - fRows[1][1] * fRows[2][0]),
                        Vector3D(fRows[0][2] * fRows[2][1] - fRows[0][1] * fRows[2][2],
                                 fRows[0][0] * fRows[2][2] - fRows[0][2] * fRows[2][0],
                                 fRows[0][1] * fRows[2][0] - fRows[0][0] * fRows[2][1]),
                        Vector3D(fRows[0][1] * fRows[1][2] - fRows[0][2] * fRows[1][1],
                                 fRows[0][2] * fRows[1][0] - fRows[0][0] * fRows[1][2],
                                 fRows[0][0] * fRows[1][1] - fRows[0][1] * fRows[1][0])
                       );

        // Transpose the matrix of cofactors
        Matrix3x3 adjudant= cofactor.transpose();

        return adjudant * (1.0f / det());
}

std::ostream& operator<<( std::ostream& aOStream, const Matrix3x3& aMatrix ){
           aOStream << "[" << aMatrix.row(0) << ","
                           << aMatrix.row(1) << ","
                           << aMatrix.row(2) << "]";
    return aOStream;
}
```

```
//
//  Polygon_PS1.cpp
//  Assignment1
//
//  Created by Ekrar Efaz on 22/3/23.
//

#include "Polygon.h"
#include "Matrix3x3.h"

float Polygon::getSignedArea() const noexcept
{
    float area = 0.0f;

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


        // handle last vertex
        if (i == fNumberOfVertices-1)
        {
            const Vector2D& firstVertex = fVertices[0];
            const Vector2D& lastVertex = fVertices[fNumberOfVertices-1];
            area += lastVertex.x() * firstVertex.y() - firstVertex.x() * lastVertex.y();
        }
        else{
            const Vector2D& currentVertex = fVertices[i];
            const Vector2D& adjacentVertex = fVertices[(i + 1)];

            area += currentVertex.x() * adjacentVertex.y() - adjacentVertex.x() * currentVertex.y();
        }
    }
    area = area * 0.5f;
    return area;
}
//float Polygon::getSignedArea() const noexcept
//{
//    float area = 0.0;
//
//    for (size_t i = 0; i < fNumberOfVertices; i++)
//    {
//        const Vector2D& vertex1 = fVertices[i];
//        const Vector2D& vertex2 = fVertices[(i + 1) % fNumberOfVertices];
//
//        float crossProduct = vertex1.x() * vertex2.y() - vertex2.x() * vertex1.y();
//        area += crossProduct;
//    }
//
//    return area / 2.0;
//}

Polygon Polygon::transform( const Matrix3x3& aMatrix ) const noexcept{

    Polygon aTransform(*this);
    for (size_t i = 0; i < fNumberOfVertices; i++){
        aTransform.fVertices[i] = static_cast<Vector2D> (aMatrix * aTransform.fVertices[i]);
    }
    return aTransform;
}
```