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: Md Redwan Ahmed Zawad **Your student ID: 103501849**

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:_____

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


Matrix3x3 Matrix3x3::operator*(const Matrix3x3& aOther)const noexcept
{
	Matrix3x3 lMat= Matrix3x3(operator*(aOther.column(0)),
		operator*(aOther.column(1)),
		operator*(aOther.column(2)));

	return lMat.transpose();

}

float Matrix3x3::det()const noexcept
{
	float lDet=0;
	size_t lIn[2];
	for (size_t i = 0; i < 3; i++)
	{
		Vector2D lVec[2];
		size_t p = 0;

		for (size_t k = 0; k < 2; k++) {
			for (size_t j = 0; j < 3; j++)
			{

				if (j != i) {
					lIn[p] = j;
					p++;
				}

			}
			lVec[k] = Vector2D(row(k+1)[lIn[0]], row(k+1)[lIn[1]]);
			p = 0;
		}
		lDet += row(0)[i] * lVec[0].cross(lVec[1])*(static_cast<float>(pow(-
1,i)));
	}

	return lDet;

}

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

Matrix3x3 Matrix3x3::inverse()const
{
	assert(det() != 0);
	float lVal[9];
		Vector2D lVec[2];
		size_t lIn[2];
		size_t k = 0;
		size_t lInd = 0;
```

```
		for (size_t iRow = 0; iRow < 3; iRow++)
		{

			for(size_t jCol=0; jCol<3;jCol++)
			{
				k = 0;
				lIn[0] = 4;
				size_t p = 0;
				for (size_t i = 0; i < 3; i++)
				{

					if (i != iRow)
					{
						if (p == 0) {
							for (size_t j = 0; j < 3; j++)
							{
								if (j != jCol && lIn[0] != j)
								{
									lIn[p] = j;
									p++;
								}
							}

						}
	lVec[k]=Vector2D(row(i)[lIn[0]],row(i)[lIn[1]]);
						k++;


					}

				}

				lVal[lInd] =
lVec[0].cross(lVec[1])*(static_cast<float>(pow(-1,iRow+jCol)));
				lInd++;


			}



		}
		Matrix3x3 lMat = Matrix3x3(Vector3D(lVal[0], lVal[1], lVal[2]),
			Vector3D(lVal[3], lVal[4], lVal[5]),
			Vector3D(lVal[6], lVal[7], lVal[8])
		);
		return lMat.transpose()* (1 / det());
}
bool Matrix3x3::hasInverse() const noexcept
{
	if (det() == 0) {
		return false;
	}
	else {
		return true;
	}
}
```

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

```cpp
#include"Polygon.h"
#include"Matrix3x3.h"
#include"Vector3D.h"

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

	for (size_t i = 0; i < getNumberOfVertices()-1; i++)
	{
		lSum+= .5f*(getVertex(i).y()+getVertex(i+1).y()) * (getVertex(i).x()-
getVertex(i+1).x());

	}
	lSum += .5f*(getVertex(getNumberOfVertices()-1).y() + getVertex(0).y()) *
(getVertex(getNumberOfVertices()-1).x() - getVertex(0).x());

	return lSum;
}

Polygon Polygon::transform(const Matrix3x3& aMatrix)const noexcept
{
	Polygon Result = *this;
	Vector3D lTrans;
	for (size_t i = 0; i < getNumberOfVertices(); i++)
	{
		lTrans = Vector3D(getVertex(i));
		lTrans = aMatrix * (lTrans);
		Result.fVertices[i] = lTrans.operator Vector2D();
	}
	return Result;
}
```