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: Luan Nguyen **Your student ID: 103812143**

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:_____

Problem 1- toString() function implementation

```cpp
//
//  Vector3D_PS1.cpp
//  ProblemSet1
//
//  Created by Luan Nguyen on 19/3/2024.
//


#include "Vector3D.h"

#include <cassert>
#include <cmath>
#include <sstream>


// Problem 1 - Implement ToString extention
std::string Vector3D::toString() const noexcept{

    // Create a string stream for constructing the string representation
    std::stringstream ss;

    //Round the components to four decimal
    ss << "[" << std::round(fBaseVector.x() * 10000.0f) / 10000.0f << ","
    << std::round(fBaseVector.y() * 10000.0f) / 10000.0f << ","
    << std::round(fW * 10000.0f) / 10000.0f << "]";

    // Convert to string
    return ss.str();
}

```

Figure 1. Problem1 - toString()

Reformat the value to round to 4 decimal and convert to string

```
Vector a: [1,2,3]
Vector b: [3.1416,3.1416,3.1416]
Vector c: [1.2346,9.8765,12435.1]
1 Test(s) completed.
Program ended with exit code: 0
```

Figure 2. Test 1 result

Problem 2: Matrices

1. Multiply 2 Matrices

ProblemSet1 〉 ProblemSet1 〉 Sources 〉 Matrix3x3_PS1 〉 Matrix3x3::det()

```cpp
//
//  Matrix3x3_PS1.cpp
//  ProblemSet1
//
//  Created by Luan Nguyen on 20/3/2024.
//

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



// Multiply 2 Matrices
Matrix3x3 Matrix3x3::operator*(const Matrix3x3& aOther) const noexcept{
    // Declare local constant rows
    const Vector3D& lrow1 = this->row(0);
    const Vector3D& lrow2 = this->row(1);
    const Vector3D& lrow3 = this->row(2);
    // Declare constant columns of the Matrix
    const Vector3D& lcolumn1 = aOther.column(0);
    const Vector3D& lcolumn2 = aOther.column(1);
    const Vector3D& lcolumn3 = aOther.column(2);

    // Create 3 Rows for the matrix
    Vector3D lRow1( lrow1.dot(lcolumn1),
                    lrow1.dot(lcolumn2),
                    lrow1.dot(lcolumn3)
                   );
    Vector3D lRow2( lrow2.dot(lcolumn1),
                    lrow2.dot(lcolumn2),
                    lrow2.dot(lcolumn3)
                    );
    Vector3D lRow3( lrow3.dot(lcolumn1),
                    lrow3.dot(lcolumn2),
                    lrow3.dot(lcolumn3)
                   );
    // Return the created matrix
    return Matrix3x3(lRow1, lRow2, lRow3);

}

```

Figure 3. Multiply Matrices

Declaring the local reference to maximize the memory using while not calling row() and column() too many times, dot() the column() of this matrix to the row() of the other to get the entry of the new Matrix and return the how 3x3Matrix

2. Determine of a Matrix

For a 3 x 3 matrix **M**, the determinate of **M** is given by

$$\begin{aligned}\det \mathrm{M} = & \quad M_{11}(M_{22}M_{33} - M_{23}M_{32}) \\ & - M_{12}(M_{21}M_{33} - M_{23}M_{31}) \\ & + M_{13}(M_{21}M_{32} - M_{22}M_{31})\end{aligned}$$

By apply this fomular, we can calculate the determine of the matrix using row(column_Index)[row_Index] to access the value

```
// Determine of a Matrix
float Matrix3x3::det() const noexcept{
    // Declare local constant rows
    const Vector3D& lrow1 = this->row(0);
    const Vector3D& lrow2 = this->row(1);
    const Vector3D& lrow3 = this->row(2);


    float result = 0.0f;
    result += lrow1[0] * (lrow2[1]*lrow3[2] - lrow2[2]*lrow3[1]);
    result -= lrow1[1] * (lrow2[0]*lrow3[2] - lrow2[2]*lrow3[0]);
    result += lrow1[2] * (lrow2[0]*lrow3[1] - lrow2[1]*lrow3[0]);

    return result;
}

```

Figure 4. Determine of a Matrix

3. Transpose the Matrix

Convert the row to column of the Matrix to transpose it in which the n x m **M** to m x n **M**

```
// Transpose the Matrix
Matrix3x3 Matrix3x3::transpose() const noexcept{

    return Matrix3x3(column(0),column(1),column(2));
}
```

Figure 5. Transpose

4. Invertibility

If the det() of the matrix is not 0 so it is invertible

```
// Check of the Matrix is invertible
bool Matrix3x3::hasInverse() const noexcept{
    if (det() == 0.0f)
        return false;
    else
        return true;
}
```

Figure 6. Invertibility

5. Inverse the matrix

For a 3 x 3 matrix **M**, the inverse matrix $\mathbf{M}^{-1}$ is given by

$$\mathrm{M}^{-1} = \frac{1}{\det \mathrm{M}} \begin{bmatrix} M_{22}M_{33} - M_{23}M_{32} & M_{13}M_{32} - M_{12}M_{33} & M_{12}M_{23} - M_{13}M_{22} \\ M_{23}M_{31} - M_{21}M_{33} & M_{11}M_{33} - M_{13}M_{31} & M_{13}M_{21} - M_{11}M_{23} \\ M_{21}M_{32} - M_{22}M_{31} & M_{12}M_{31} - M_{11}M_{32} & M_{11}M_{22} - M_{12}M_{21} \end{bmatrix}$$

Similarly to det() access the matrix using row() and then multipy it with 1/det()

```
// Inverse of a matrix

Matrix3x3 Matrix3x3::inverse() const noexcept{
    const float& ldet = this->det();
    assert(ldet!=0);
    // Declare local constant rows
    const Vector3D& lrow1 = this->row(0);
    const Vector3D& lrow2 = this->row(1);
    const Vector3D& lrow3 = this->row(2);

    Vector3D lRow1 ( lrow2[1] * lrow3[2] - lrow2[2] * lrow3[1],
                     lrow1[2] * lrow3[1] - lrow1[1] * lrow3[2],
                     lrow1[1] * lrow2[2] - lrow1[2] * lrow2[1]);

    Vector3D lRow2 ( lrow2[2] * lrow3[0] - lrow2[0] * lrow3[2],
                     lrow1[0] * lrow3[2] - lrow1[2] * lrow3[0],
                     lrow1[2] * lrow2[0] - lrow1[0] * lrow2[2]);

    Vector3D lRow3 ( lrow2[0] * lrow3[1] - lrow2[1] * lrow3[0],
                     lrow1[1] * lrow3[0] - lrow1[0] * lrow3[1],
                     lrow1[0] * lrow2[1] - lrow1[1] * lrow2[0]);

    return Matrix3x3(lRow1, lRow2, lRow3)* (1/ldet);

    }
```

Figure 7. Inverse()

6. Apply toString() to the matrix

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

Figure 8. toString()

Problem 3: Polygon

1. Get the area

By using given the trapezoid formula for polygon implement the function to calculate the area of the polygon

```
//
//  Created by Luan Nguyen on 20/3/2024.
//

#include <stdio.h>
#include "Polygon.h"



float Polygon::getSignedArea() const noexcept {
    float result = 0.0f; // Initialize the result variable to store the signed area
    for (int i = 0; i < fNumberOfVertices; i++) {
        int j = (i + 1) % fNumberOfVertices; // Get the index of the next vertex using modulus to handle wrapping
        // Calculate the area of the trapezoid formed by the current edge and the x-axis
        float lArea = (fVertices[i].x() + fVertices[j].x()) * (fVertices[j].y() - fVertices[i].y()) / 2.0f;
        // Add the local variable area to the result
        result += lArea;
    }
    return result; // Return the signed area of the polygon
}
```

Figure 9. Calculate the area

2. Transform the Polygon

   By applying a 3x3 transformation to each vertex

```
// Transform the current polygon by applying a 3x3 transformation matrix
// to each vertex, resulting in a new transformed polygon.
Polygon Polygon::transform(const Matrix3x3& aMatrix) const noexcept {
    // Create a copy of the current polygon
    Polygon result = *this;

    // Iterate through each vertex of the polygon
    for (int i = 0; i < fNumberOfVertices; i++) {
        // Transform the current vertex using the transformation matrix
        // and store the result in a temporary Vector3D object.
        Vector3D lTempVec = Vector3D(aMatrix * fVertices[i]);

        // Convert the resulting 3D vector to a 2D vector by discarding the z-coordinate,
        // and assign it to the corresponding vertex of the transformed polygon.
        result.fVertices[i] = static_cast<Vector2D>(lTempVec);
    }

    // Return the transformed polygon
    return result;
}
```

Figure 10. Transform the Polygon