

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: _____ **Your student ID:** _____

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: _____

Problem Set 1: Solution Design in C++

Problem 1

Start with the solution of tutorial 3 in which we implemented the classes `Vector3D` and `Matrix3x3` to perform vector transformations in 2D.

In this task, we wish to extend the definition of class `Matrix3x3` with some additional matrix operations. In particular, we extend class `Matrix3x3` with

- Matrix multiplication:

Two matrices **F** and **G** can be multiplied, provided that the number of columns in **F** is equal to the number of rows in **G**. If **F** is $n \times m$ matrix and **G** is an $m \times p$ matrix, then the product **FG** is an $n \times p$ matrix whose (i, j) entry is given by

$$(\mathbf{FG})_{ij} = \sum_{k=1}^m F_{ik} G_{kj}$$

The entry $(\mathbf{FG})_{ij}$ is the dot product of $\text{row}(\mathbf{F}, i)$ and $\text{column}(\mathbf{G}, j)$.

In the implementation, every row and column should ideally be accessed from the corresponding matrices at most once to avoid repeated copies. Computing the result does not require loops.

- Determinate of a matrix:

The determinate is a scalar value that is a function of the entities of a square matrix. It characterizes some properties of a square matrix, for example, if the matrix is invertible or if the matrix is a rotation matrix.

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

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

In the implementation, every row should ideally be accessed only once to avoid repeated copies. The row vectors are of type `Vector3D` which provides an index operator to access the corresponding column entry.

- The transpose of a matrix:

The transpose of an $n \times m$ matrix **M**, denoted by \mathbf{M}^T , is an $m \times n$ matrix for which the (i, j) entry is equal to M_{ji} . For

$$\mathbf{M}_{3 \times 3} = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix} \quad \text{the transpose is} \quad \mathbf{M}_{3 \times 3}^T = \begin{bmatrix} M_{11} & M_{21} & M_{31} \\ M_{12} & M_{22} & M_{32} \\ M_{13} & M_{23} & M_{33} \end{bmatrix}.$$

The implementation should utilize column vectors to easily construct the transpose of a matrix.

- The inverse of a matrix:

The inverse of a matrix, if it exists, allows us to represent division of matrices, a concept that is not defined for matrices. Technically, the inverse of a matrix is a multidimensional generalization of the concept of reciprocal of a number: the product of a number and its reciprocal is 1. The product of a matrix \mathbf{M} with its inverse \mathbf{M}^{-1} is the identity matrix \mathbf{I} : $\mathbf{M}\mathbf{M}^{-1} = \mathbf{I}$.

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

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

The implementation would need to compute the determinate of \mathbf{M} and verify that it is not zero. The calculation of the matrix values requires access to the row vectors of \mathbf{M} which should ideally be accessed only once to avoid repeated copies.

- A test whether a matrix \mathbf{M} is invertible:

A matrix is invertible if its determinant is not zero.

To accommodate these operations, class `Matrix3x3` is extended as follows

```
#pragma once

#include <ostream>

#include "Vector3D.h"

class Matrix3x3
{
private:
    Vector3D fRows[3];

public:
    Matrix3x3() noexcept;
    Matrix3x3( const Vector3D& aRow1, const Vector3D& aRow2, const Vector3D& aRow3 ) noexcept;

    Matrix3x3 operator*( const float aScalar ) const noexcept;
    Matrix3x3 operator+( const Matrix3x3& aOther ) const noexcept;

    Vector3D operator*( const Vector3D& aVector ) const noexcept;

    static Matrix3x3 scale( const float aX = 1.0f, const float aY = 1.0f );
    static Matrix3x3 translate( const float aX = 0.0f, const float aY = 0.0f );
    static Matrix3x3 rotate( const float aAngleInDegree = 0.0f );

    const Vector3D row( size_t aRowIndex ) const;
    const Vector3D column( size_t aColumnIndex ) const;

    // Problem Set 1 features

    Matrix3x3 operator*( const Matrix3x3& aOther ) const noexcept;

    float det() const noexcept;
    bool hasInverse() const noexcept;

    Matrix3x3 transpose() const noexcept;
    Matrix3x3 inverse() const;

    friend std::ostream& operator<<( std::ostream& aOStream, const Matrix3x3& aMatrix );
};
```

In addition to the new matrix operations, we also wish to define an output operator for 3 x 3 matrices. We can rely of the output operator for class `Vector3D` for this purpose. Its implementation in this problem set has been adjusted to emit all three vector components, that is, the statement `std::cout << Vector3D(1.0f, 2.0f, 1.0f);` yields `[1,2,3]` as output.

Do not edit the provided files. To implement the required features, create a new source file, say `Matrix3x3_PS1.cpp`, and define the new features here. It is not strictly required, but it helps to separate the definitions from the provided code. You have to include `Matrix3x3.h` in the new source for the code to compile.

The file `Main.cpp` contains a test function to check your implementation of the new matrix features. The code sequence

```
Matrix3x3 M ( Vector3D( 2.0f, 3.0f, 8.0f ),
               Vector3D( 6.0f, 0.0f, -3.0f ),
               Vector3D( -1.0f, 3.0f, 2.0f ) );

std::cout << "Test matrix M:" << std::endl;
std::cout << M << std::endl;

// test multiplication

std::cout << "M * M = " << std::endl;
std::cout << M * M << std::endl;

// test determinate

std::cout << "det M = " << M.det() << std::endl;

// test hasInverse

std::cout << "Has M a inverse? " << (M.hasInverse() ? "Yes" : "No") << std::endl;

// test transpose
std::cout << "transpose of M:" << std::endl;
std::cout << M.transpose() << std::endl;

// test inverse
std::cout << "inverse of M:" << std::endl;
std::cout << M.inverse() << std::endl;

std::cout << "inverse of M * 45:" << std::endl;
std::cout << M.inverse() * 45.0f << std::endl;
```

Should produce the following output

```
Test matrix M:
[[2,3,8],[6,0,-3],[-1,3,2]]
M * M =
[[14,30,23],[15,9,42],[14,3,-13]]
det M = 135
Has M a inverse? Yes
transpose of M:
[[2,6,-1],[3,0,3],[8,-3,2]]
inverse of M:
[[0.0666667,0.133333,-0.0666667],[-0.0666667,0.0888889,0.4],[0.133333,-0.0666667,-0.133333]]
inverse of M * 45:
[[3,6,-3],[-3,4,18],[6,-3,-6]]
```

Floating point values are printed with standard precision for type `float`.

Problem 2

Carl Friedrich Gauss and Carl Gustav Jacob Jacobi invented the trapezoid formula to calculate the area of a trapezoid in the 18th century. We can use the trapezoid formula to determine the both the area of a polygon and the orientation of the vertices of the polygon. If the vertices are ordered in clockwise order, then the area is negative. If the vertices are arranged in counterclockwise order, then the area is positive as shown in Figure 1. In computer graphics, we use the orientation of a polygon to determine whether the polygon faces the viewer or not. The latter allows for a process called *back face culling* that means, we do not have to draw the polygon. Only if the polygon faces the viewer (even partly) do we have to draw it.

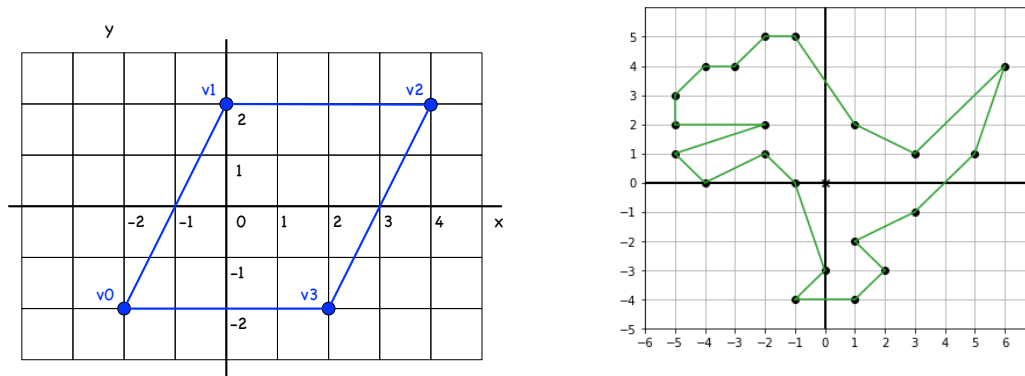


Figure 1: Parallelogram with signed area -16, dinosaur with signed area 38.5.

The trapezoid formula is given below

$$A = \frac{1}{2} \sum_{i=1}^n (y_i + y_{i \oplus 1})(x_i - x_{i \oplus 1})$$

It sums up a sequence of oriented areas of trapezoids with vertices v_i and v_{i+1} as one of its four edges. Please note that we need to connect the last with the first vertex. In the trapezoid formula, this is expressed by the modulus operator (i.e., $i \oplus 1$). Naturally, it is always possible to avoid the use of the expensive modulus operation by separating the logic into two parts (i.e., we explicitly define the connection between the last and the first vertex).

In order to add support for the signed area calculation, start with the solution of tutorial 2 (see Canvas) and use the extended specification of class `Polygon` is shown below. Please note that the extended version also includes a method to transform a given polygon. Polygon transformation requires a 3 x 3 matrix which is multiplied with every vertex of the polygon.

```

#pragma once

#include "Vector2D.h"

#define MAX_VERTICES 30

class Polygon
{
private:
    Vector2D fVertices[MAX_VERTICES];
    size_t fNumberOfVertices;

public:
    Polygon() noexcept;

    void readData( std::istream& aIStream );

    size_t getNumberOfVertices() const noexcept;
    const Vector2D& getVertex( size_t aIndex ) const;
    float getPerimeter() const noexcept;

    Polygon scale( float aScalar ) const noexcept;

    // Problem Set 1 extension

    float getSignedArea() const noexcept;

    Polygon transform( const Matrix3x3& aMatrix ) const noexcept;
};

```

As in 0, do not edit the provided files. Rather create a new source file, called `Polygon_PS1.cpp`, to implement the new methods `getSignedArea()` and `transform()`.

There are two sample files that you can use to test your solution: `Parallelogram.txt` and `Data.txt`. The file `Main.cpp` contains a test function, `testProblem2()`, to verify your implementation. For the input file `Parallelogram.txt` the test function outputs

```

Signed area: -16
Signed area of rotated polygon: -16
Polygon transformation successful.

```

The vertices in the parallelogram are arranged in clockwise order. The area of a polygon does not change when it is simply rotated around the origin.

For the input file `Data.txt` the test function outputs

```

Signed area: 38.5
Signed area of rotated polygon: 38.5
Polygon transformation successful.

```

The vertices in the T-Rex polygon are arranged in counterclockwise order.

Submission deadline: Monday, March 27, 2023, 10:30.

Submission procedure: Follow the instruction on Canvas. Submit electronically PDF of printed code for `Matrix3x3_PS1.cpp` and `PolygonPS1.cpp`. Upload the of `Matrix3x3_PS1.cpp` and `PolygonPS1.cpp`.

The sources need to compile in the presents of the solution artifacts provided on Canvas.