PolygonPS1

```
//
//  PolygonPS1.cpp
//  problem asm1
//
//  Created by Vu Duc Tran on 19/3/2024.
//
#define _USE_MATH_DEFINES     // must be defined before any #include
#include "Matrix3x3.h"
#include "Vector3D.h"
#include <cassert>
#include <cmath>
#include <sstream>
#include <iomanip>
#include "Polygon.h"

using namespace std;

Polygon::Polygon() noexcept :
    fNumberOfVertices()
{}

void Polygon::readData( std::istream& aIStream )
{
    // read input file containing 2D vector data
    // if no data can be read, then exit loop
    while ( aIStream >> fVertices[fNumberOfVertices] )
    {
        fNumberOfVertices++;
    }
}

size_t Polygon::getNumberOfVertices() const noexcept
{
    return fNumberOfVertices;
}

const Vector2D& Polygon::getVertex( size_t aIndex ) const
{
    assert( aIndex < fNumberOfVertices );

    return fVertices[aIndex];
}

float Polygon::getPerimeter() const noexcept
{
    float Result = 0.0f;

    // There have to be at least three vertices
    if ( fNumberOfVertices > 2 )
    {
        // solution without modulus and explicit temporary variables
        for ( size_t i = 1; i < fNumberOfVertices; i++ )
        {
            Result += (fVertices[i] - fVertices[i - 1]).length();
        }
        Result += (fVertices[0] - fVertices[fNumberOfVertices - 1]).length();
    }

    return Result;
}

Polygon Polygon::scale( float aScalar ) const noexcept
{
    Polygon Result = *this;

    for ( size_t i = 0; i < fNumberOfVertices; i++ )
    {
        Result.fVertices[i] = fVertices[i] * aScalar;
    }

    return Result;
}

float Polygon::getSignedArea() const noexcept {
    float Result = (fVertices[fNumberOfVertices - 1].x() - fVertices[0].x()) *
(fVertices[fNumberOfVertices - 1].y() + fVertices[0].y());

    for (int i = 0; i < fNumberOfVertices - 1; ++i) {
        Result += (fVertices[i].x() - fVertices[i + 1].x()) *
(fVertices[i].y() + fVertices[i + 1].y());
    }

    Result /= 2;

    return Result;
}

Polygon Polygon::transform(const Matrix3x3& aMatrix) const noexcept {
    Polygon Result = *this;
    for (int i = 0; i < fNumberOfVertices; ++i) {
        Vector3D temp = Vector3D(fVertices[i]);
        Vector3D dot_product = aMatrix * temp;
        Result.fVertices[i] = Vector2D(dot_product);
    }
    return Result;
}
```