```
//
//  Matrix3x3_PS1.cpp
//  problem asm1
//
//  Created by Vu Duc Tran on 17/3/2024.
//
#define _USE_MATH_DEFINES     // must be defined before any #include
#include "Matrix3x3.h"
#include "Vector3D.h"
#include <cassert>
#include <cmath>
#include <sstream>
#include <iomanip>

using namespace std;

Matrix3x3 Matrix3x3::operator*( const Matrix3x3& aOther ) const noexcept
{
    Vector3D rows[3];
    for (int i = 0; i < 3; ++i)
    {
        float val[3];
        val[0] = fRows[i][0] * aOther.row(0).x() + fRows[i][1] *
aOther.row(1).x() + fRows[i][2] * aOther.row(2).x();
        val[1] = fRows[i][0] * aOther.row(0).y() + fRows[i][1] *
aOther.row(1).y() + fRows[i][2] * aOther.row(2).y();
        val[2] = fRows[i][0] * aOther.row(0).w() + fRows[i][1] *
aOther.row(1).w() + fRows[i][2] * aOther.row(2).w();
        rows[i] = Vector3D(val[0], val[1], val[2]);
    }

    return Matrix3x3(rows[0], rows[1], rows[2]);
}

float Matrix3x3::det() const noexcept
{
    float res = 0;
    int pos_index[3][3] = {{0, 1, 2}, {1, 2, 0}, {2, 0, 1}};
    int neg_index[3][3] = {{0, 2, 1}, {1, 0, 2}, {2, 1, 0}};
    for (int i = 0; i < 3; ++i)
    {
        res += fRows[0][pos_index[i][0]] * fRows[1][pos_index[i][1]] *
fRows[2][pos_index[i][2]]
               - fRows[0][neg_index[i][0]] * fRows[1][neg_index[i][1]] *
fRows[2][neg_index[i][2]];
    }
    return res;
}


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

```
Matrix3x3 Matrix3x3::transpose() const noexcept
{
    if (!hasInverse())
    {
        return Matrix3x3();
    }
    Vector3D result[3];
    for (int i = 0; i < 3; ++i)
    {
        result[i] = Vector3D(fRows[0][i], row(1)[i], row(2)[i]);
    }
    return Matrix3x3(result[0], result[1], result[2]);
}

Matrix3x3 Matrix3x3::inverse() const noexcept
{
    Vector3D rows[3];
    int template_index[3][2] = {{1, 2}, {0, 2}, {0, 1}};
    for (int i = 0; i < 3; ++i)
    {
        float tmp_value[3];
        for (int j = 0; j < 3; ++j)
        {
            //template_index[i] = row index chosen to calculate the current
position
            //template_index[j] = column index chosen to calculate the
current position
            tmp_value[j] = fRows[template_index[i][0]][template_index[j][0]]
* fRows[template_index[i][1]][template_index[j][1]] -
fRows[template_index[i][0]][template_index[j][1]] *
fRows[template_index[i][1]][template_index[j][0]];

            if ((i + j) % 2 != 0) {
                tmp_value[j] *= -1;
            }
        }
        rows[i] = Vector3D(tmp_value[0], tmp_value[1], tmp_value[2]);
    }
    return Matrix3x3(rows[0], rows[1], rows[2]).transpose() * (1 / det());
}

//Write
std::ostream& operator<<( std::ostream& aOStream, const Matrix3x3& aMatrix )
{
    for (int i = 0; i < 3; ++i)
    {
        if (i != 2)
        {
            aOStream << "[" << std::round( aMatrix.row(i).x() * 10000.0f) /
10000.0f << "," << std::round( aMatrix.row(i).y() * 10000.0f) / 10000.0f <<
"," << std::round( aMatrix.row(i).w() * 10000.0f) / 10000.0f << "],";
        }
        else
        {
```

```
            aOStream << "[" << std::round( aMatrix.row(i).x() * 10000.0f) /
10000.0f << "," << std::round( aMatrix.row(i).y() * 10000.0f) / 10000.0f <<
"," << std::round( aMatrix.row(i).w() * 10000.0f) / 10000.0f << "]";
        }
    }
    return aOStream;
}
```