```
//
//  Matrix3x3_PS1.cpp
//  problemset1
//
//  Created by Xinzhe Yu on 23/3/2025.
//

#include "Matrix3x3.h"
#include <cassert>
#include <sstream>


bool Matrix3x3::operator==(const Matrix3x3 &aOther) const noexcept{
    const Matrix3x3& M = *this;
    return (M[0] == aOther.fRows[0]) && (M[1] == aOther.fRows[1]) && (M[2]
     == aOther.fRows[2]);
}

Matrix3x3 Matrix3x3::operator*(const Matrix3x3& aOther) const noexcept {
    // the multiplication of 2 3x3 matrix can be calculated by using M[x]
     to get each horizontal vector of the first matrix and using column(x)
     to get each vertical vector of the second matrix, then using dot() to
     get the new x y w for the new vector inside of the result matrix
    const Matrix3x3& M = *this;
    return Matrix3x3(
        Vector3D(M[0].dot(aOther.column(0)), M[0].dot(aOther.column(1)),
         M[0].dot(aOther.column(2))),
        Vector3D(M[1].dot(aOther.column(0)), M[1].dot(aOther.column(1)),
         M[1].dot(aOther.column(2))),
        Vector3D(M[2].dot(aOther.column(0)), M[2].dot(aOther.column(1)),
         M[2].dot(aOther.column(2)))
    );
}

Matrix3x3 Matrix3x3::transpose() const noexcept{
    const Matrix3x3& M = *this;
    return Matrix3x3(M.column(0), M.column(1), M.column(2));
}

float Matrix3x3::det() const noexcept {
    const Matrix3x3& M = *this;
    return M[0].x() * (M[1].y() * M[2].w() - M[1].w() * M[2].y()) -
    M[0].y() * (M[1].x() * M[2].w() - M[1].w() * M[2].x()) +
    M[0].w() * (M[1].x() * M[2].y() - M[1].y() * M[2].x());
}

bool Matrix3x3::hasInverse() const noexcept{
    // check whether the determination of this matrix is more than the
     smallest difference with 0
    return std::fabs(det()) > std::numeric_limits<float>::epsilon();
}

Matrix3x3 Matrix3x3::inverse() const noexcept {
    assert(hasInverse());
    const Matrix3x3& M = *this;
```

```
    // the inverse of a 3x3 matrix is equal to the inverse of its det
     multiple with Adjugate Matrix
    // the inverse of its det can be calculated directly by det()
    float invDet = 1.0f / det();
    return Matrix3x3(
        Vector3D((M[1][1]*M[2][2]-M[1][2]*M[2][1])*invDet,
         (M[0][2]*M[2][1]-M[0][1]*M[2][2])*invDet,
         (M[0][1]*M[1][2]-M[0][2]*M[1][1])*invDet),
        Vector3D((M[1][2]*M[2][0]-M[1][0]*M[2][2])*invDet,
         (M[0][0]*M[2][2]-M[0][2]*M[2][0])*invDet,
         (M[0][2]*M[1][0]-M[0][0]*M[1][2])*invDet),
        Vector3D((M[1][0]*M[2][1]-M[1][1]*M[2][0])*invDet,
         (M[0][1]*M[2][0]-M[0][0]*M[2][1])*invDet,
         (M[0][0]*M[1][1]-M[0][1]*M[1][0])*invDet)
    );
}

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