```cpp
// COS30008, tutorial 3, 2023

#define _USE_MATH_DEFINES // must be defined before any #include

#include "Matrix3x3_1.h"

#include <cassert>
#include <cmath>

Matrix3x3::Matrix3x3() noexcept {
    fRows[0] = Vector3D(1.0f, 0.0f, 0.0f);
    fRows[1] = Vector3D(0.0f, 1.0f, 0.0f);
    fRows[2] = Vector3D(0.0f, 0.0f, 1.0f);
}

Matrix3x3::Matrix3x3(const Vector3D& aRow1, const Vector3D& aRow2,
    const Vector3D& aRow3) noexcept {
    fRows[0] = aRow1;
    fRows[1] = aRow2;
    fRows[2] = aRow3;
}

const Vector3D Matrix3x3::row(size_t aRowIndex) const {
    assert(aRowIndex < 3);

    return fRows[aRowIndex];
}

const Vector3D Matrix3x3::column(size_t aColumnIndex) const {
    assert(aColumnIndex < 3);

    return Vector3D(row(0)[aColumnIndex], row(1)[aColumnIndex],
        row(2)[aColumnIndex]);
}

Matrix3x3 Matrix3x3::operator*(const float aScalar) const noexcept {
    return Matrix3x3(row(0) * aScalar, row(1) * aScalar, row(2) * aScalar);
}
//
//SOLUTION START
//
Matrix3x3 Matrix3x3::operator*(const Matrix3x3& aOther) const noexcept
{
    return Matrix3x3(row(0) * aOther.column(0),
        row(1) * aOther.column(1),
        row(2) * aOther.column(2));
}

float Matrix3x3::det() const noexcept {
    const float a = fRows[0][0], b = fRows[0][1], c = fRows[0][2];
    const float d = fRows[1][0], e = fRows[1][1], f = fRows[1][2];
    const float g = fRows[2][0], h = fRows[2][1], i = fRows[2][2];
    const float detA = e * i - f * h;
    const float detB = d * i - f * g;
    const float detC = d * h - e * g;

    return a * detA - b * detB + c * detC;
}

Matrix3x3 Matrix3x3::transpose() const noexcept {
    return Matrix3x3(Vector3D(fRows[0].x(), fRows[1].x(), fRows[2].x()),
        Vector3D(fRows[0].y(), fRows[1].y(), fRows[2].y()),
        Vector3D(fRows[0].w(), fRows[1].w(), fRows[2].w()));
}

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

Matrix3x3 Matrix3x3::inverse() const {
    float detM = det();
    assert(detM != 0);

    float invDetM = 1.0f / detM;

    Vector3D row0 = fRows[0];
    Vector3D row1 = fRows[1];
    Vector3D row2 = fRows[2];

    float M11 = row1.y() * row2.w() - row1.w() * row2.y();
    float M12 = row0.w() * row2.y() - row0.y() * row2.w();
    float M13 = row0.y() * row1.w() - row0.w() * row1.y();

    float M21 = row1.w() * row2.x() - row1.x() * row2.w();
    float M22 = row0.x() * row2.w() - row0.w() * row2.x();
    float M23 = row0.w() * row1.x() - row0.x() * row1.w();

    float M31 = row1.x() * row2.y() - row1.y() * row2.x();
    float M32 = row0.y() * row2.x() - row0.x() * row2.y();
    float M33 = row0.x() * row1.y() - row0.y() * row1.x();

    Matrix3x3 invM;
    invM.fRows[0] = Vector3D(M11, M12, M13) * invDetM;
    invM.fRows[1] = Vector3D(M21, M22, M23) * invDetM;
    invM.fRows[2] = Vector3D(M31, M32, M33) * invDetM;

    return invM;
}

Matrix3x3 Matrix3x3::operator+(const Matrix3x3& aOther) const noexcept {
    return Matrix3x3(row(0) + aOther.row(0), row(1) + aOther.row(1),
        row(2) + aOther.row(2));
}

Vector3D Matrix3x3::operator*(const Vector3D& aVector) const noexcept {
    return Vector3D(row(0).dot(aVector), row(1).dot(aVector),
        row(2).dot(aVector));
}

Matrix3x3 Matrix3x3::scale(const float aX, const float aY) {
    return Matrix3x3(Vector3D(aX, 0.0f, 0.0f), Vector3D(0.0f, aY, 0.0f),
        Vector3D(0.0f, 0.0f, 1.0f));
}

Matrix3x3 Matrix3x3::translate(const float aX, const float aY) {
    return Matrix3x3(Vector3D(1.0f, 0.0f, aX), Vector3D(0.0f, 1.0f, aY),
        Vector3D(0.0f, 0.0f, 1.0f));
}

Matrix3x3 Matrix3x3::rotate(const float aAngleInDegree) {
    float lRadTheta = aAngleInDegree * static_cast<float>(M_PI) / 180.0f;

    float lSinTheta = std::sin(lRadTheta);
    float lCosTheta = std::cos(lRadTheta);

    return Matrix3x3(Vector3D(lCosTheta, -lSinTheta, 0.0f),
        Vector3D(lSinTheta, lCosTheta, 0.0f),
        Vector3D(0.0f, 0.0f, 1.0f));
}

std::ostream& operator<<(std::ostream& os, const Matrix3x3& matrix) {
    os << "[[" << matrix.fRows[0][0] << ", " << matrix.fRows[0][1] << ", "
        << matrix.fRows[0][2] << "], ";
    os << "[" << matrix.fRows[1][0] << ", " << matrix.fRows[1][1] << ", "
        << matrix.fRows[1][2] << "], ";
    os << "[" << matrix.fRows[2][0] << ", " << matrix.fRows[2][1] << ", "
        << matrix.fRows[2][2] << "]]";
    return os;
}
//
//SOLUTION END
//
```