

```

#define _USE_MATH_DEFINES           // must be defined before any #include
#include "Matrix3x3.h"
#include <cmath>
#include <sstream>
#include <iomanip>

```

```

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

```

```

float Matrix3x3::det() const noexcept
{
    const Vector3D& aRow1 = row(0);
    const Vector3D& aRow2 = row(1);
    const Vector3D& aRow3 = row(2);

    return aRow1[0] * (aRow2[1] * aRow3[2] - aRow2[2] * aRow3[1]) -
        aRow1[1] * (aRow2[0] * aRow3[2] - aRow2[2] * aRow3[0]) +
        aRow1[2] * (aRow2[0] * aRow3[1] - aRow2[1] * aRow3[0]);
}

```

```

Matrix3x3 Matrix3x3::transpose() const noexcept
{
    return Matrix3x3(
        Vector3D(row(0)[0], row(1)[0], row(2)[0]),
        Vector3D(row(0)[1], row(1)[1], row(2)[1]),
        Vector3D(row(0)[2], row(1)[2], row(2)[2])
    );
}

```

```

bool Matrix3x3::hasInverse() const noexcept
{
    // Matrix is invertible if the determinant is non-zero
    return det() != 0.0f;
}

```

```

Matrix3x3 Matrix3x3::inverse() const noexcept
{
    // Calculate determinant
    float determinant = det();

    // Throw error message back when the matrix is not invertible in tester
    if (determinant == 0.0f)
        throw std::logic_error("Matrix is not invertible");

    // Calculate inverse
    float invDet = 1.0f / determinant;

    return Matrix3x3(
        Vector3D(
            (fRows[1][1] * fRows[2][2] - fRows[1][2] * fRows[2][1]) * invDet,
            (fRows[0][2] * fRows[2][1] - fRows[0][1] * fRows[2][2]) * invDet,
            (fRows[0][1] * fRows[1][2] - fRows[0][2] * fRows[1][1]) * invDet
        ),
        Vector3D(
            (fRows[1][2] * fRows[2][0] - fRows[1][0] * fRows[2][2]) * invDet,
            (fRows[0][0] * fRows[2][2] - fRows[0][2] * fRows[2][0]) * invDet,
            (fRows[0][2] * fRows[1][0] - fRows[0][0] * fRows[1][2]) * invDet
        ),

```

```

        Vector3D(
            (fRows[1][0] * fRows[2][1] - fRows[1][1] * fRows[2][0]) * invDet,
            (fRows[0][1] * fRows[2][0] - fRows[0][0] * fRows[2][1]) * invDet,
            (fRows[0][0] * fRows[1][1] - fRows[0][1] * fRows[1][0]) * invDet
        )
    };
}

// Non-integer values return with 4 decimal places.
std::ostream& operator<<(std::ostream& aOStream, const Vector3D& aVector) {
    aOStream << "(" << std::roundf(10000.0 * aVector.x()) / 10000.0 << ", "
        << std::roundf(10000.0 * aVector.y()) / 10000.0 << ", "
        << std::roundf(10000.0 * aVector.w()) / 10000.0 << ")";
    return aOStream;
}

// Implementation of operator<< for Matrix3x3
std::ostream& operator<<(std::ostream& aOStream, const Matrix3x3& aMatrix) {
    aOStream << "[" << aMatrix.row(0) << ", " << aMatrix.row(1) << ", " << aMatrix.row(2) <<
    "]"<< "\n";
    return aOStream;
}

```