```cpp
#include "Matrix3x3.h"
#include <cmath>
#include <cassert>

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

}

float Matrix3x3::det() const noexcept {
    return fRows[0][0] * (fRows[1][1] * fRows[2][2] - fRows[1][2] * fRows[2][1]) -
        fRows[0][1] * (fRows[1][0] * fRows[2][2] - fRows[1][2] * fRows[2][0]) +
        fRows[0][2] * (fRows[1][0] * fRows[2][1] - fRows[1][1] * fRows[2][0]);
}

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

bool Matrix3x3::hasInverse() const noexcept {
    return det() != 0.0f;
}

Matrix3x3 Matrix3x3::inverse() const noexcept {
    // Compute the determinant of the matrix
    float lDetValue = det();

    // Check if the determinant is zero
    assert(lDetValue != 0.0f);

    // Calculate the inverse matrix using the determined determinant
    float lInvDet = 1.0f / lDetValue;

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

```
                lInvDet),
46
47          Vector3D((fRows[1][2] * fRows[2][0] - fRows[1][0] * fRows[2][2])
               * lInvDet,
48              (fRows[0][0] * fRows[2][2] - fRows[0][2] * fRows[2][0]) *
                  lInvDet,
49              (fRows[0][2] * fRows[1][0] - fRows[0][0] * fRows[1][2]) *
                  lInvDet),
50
51          Vector3D((fRows[1][0] * fRows[2][1] - fRows[1][1] * fRows[2][0])
               * lInvDet,
52              (fRows[0][1] * fRows[2][0] - fRows[0][0] * fRows[2][1]) *
                  lInvDet,
53              (fRows[0][0] * fRows[1][1] - fRows[0][1] * fRows[1][0]) *
                  lInvDet)
54      );
55  }
56
57  std::ostream& operator<<(std::ostream& os, const Matrix3x3& matrix) {
58      os << "[";
59      for (int i = 0; i < 3; ++i) {
60          os << matrix.fRows[i].toString();
61          if (i < 2) {
62              os << ",";
63          }
64      }
65      os << "]";
66      return os;
67  }
68
```