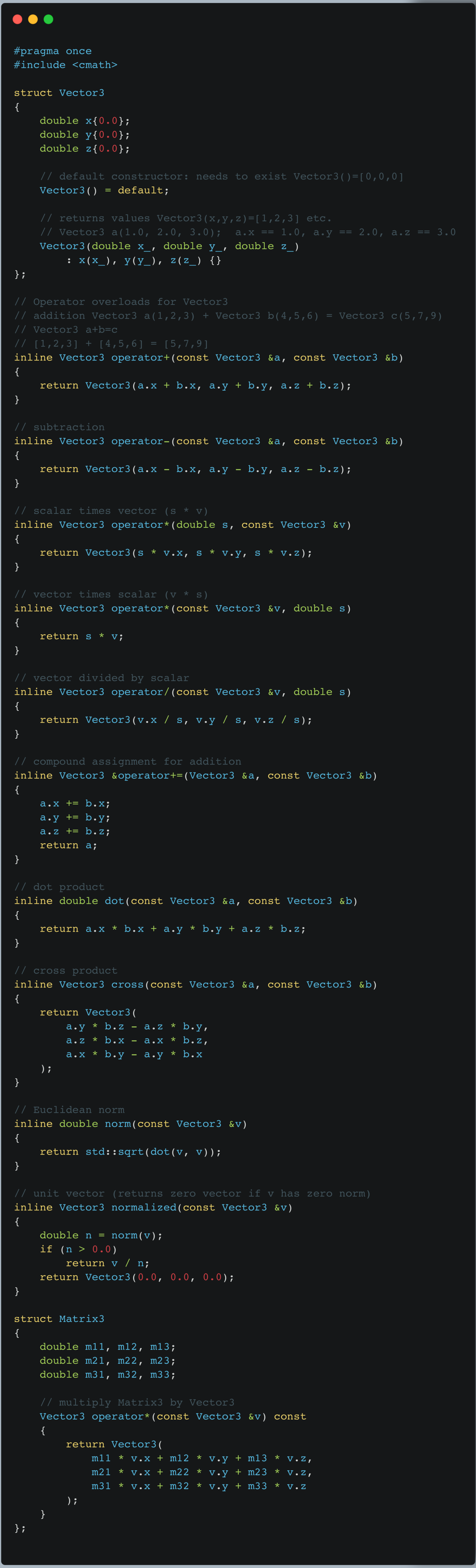

```cpp
#pragma once
#include <cmath>

struct Vector3
{
    double x{0.0};
    double y{0.0};
    double z{0.0};

    // default constructor: needs to exist Vector3()=[0,0,0]
    Vector3() = default;

    // returns values Vector3(x,y,z)=[1,2,3] etc.
    // Vector3 a(1.0, 2.0, 3.0);  a.x == 1.0, a.y == 2.0, a.z == 3.0
    Vector3(double x_, double y_, double z_)
        : x(x_), y(y_), z(z_) {}
};

// Operator overloads for Vector3
// addition Vector3 a(1,2,3) + Vector3 b(4,5,6) = Vector3 c(5,7,9)
// Vector3 a+b=c
// [1,2,3] + [4,5,6] = [5,7,9]
inline Vector3 operator+(const Vector3 &a, const Vector3 &b)
{
    return Vector3(a.x + b.x, a.y + b.y, a.z + b.z);
}

// subtraction
inline Vector3 operator-(const Vector3 &a, const Vector3 &b)
{
    return Vector3(a.x - b.x, a.y - b.y, a.z - b.z);
}

// scalar times vector (s * v)
inline Vector3 operator*(double s, const Vector3 &v)
{
    return Vector3(s * v.x, s * v.y, s * v.z);
}

// vector times scalar (v * s)
inline Vector3 operator*(const Vector3 &v, double s)
{
    return s * v;
}

// vector divided by scalar
inline Vector3 operator/(const Vector3 &v, double s)
{
    return Vector3(v.x / s, v.y / s, v.z / s);
}

// compound assignment for addition
inline Vector3 &operator+=(Vector3 &a, const Vector3 &b)
{
    a.x += b.x;
    a.y += b.y;
    a.z += b.z;
    return a;
}

// dot product
inline double dot(const Vector3 &a, const Vector3 &b)
{
    return a.x * b.x + a.y * b.y + a.z * b.z;
}

// cross product
inline Vector3 cross(const Vector3 &a, const Vector3 &b)
{
    return Vector3(
        a.y * b.z - a.z * b.y,
        a.z * b.x - a.x * b.z,
        a.x * b.y - a.y * b.x
    );
}

// Euclidean norm
inline double norm(const Vector3 &v)
{
    return std::sqrt(dot(v, v));
}

// unit vector (returns zero vector if v has zero norm)
inline Vector3 normalized(const Vector3 &v)
{
    double n = norm(v);
    if (n > 0.0)
        return v / n;
    return Vector3(0.0, 0.0, 0.0);
}

struct Matrix3
{
    double m11, m12, m13;
    double m21, m22, m23;
    double m31, m32, m33;

    // multiply Matrix3 by Vector3
    Vector3 operator*(const Vector3 &v) const
    {
        return Vector3(
            m11 * v.x + m12 * v.y + m13 * v.z,
            m21 * v.x + m22 * v.y + m23 * v.z,
            m31 * v.x + m32 * v.y + m33 * v.z
        );
    }
};
```