

```

#include <iostream>
#include <iomanip>
using namespace std;

// Klassendefinition
class Matrix_3x3 {
private:
    int m[3][3];

public:
    Matrix_3x3();
    void ausgabe() const;
    void einlesen();
    void set(int value, int i, int j);
    int get(int i, int j) const;
    void addieren(const Matrix_3x3 & ref_m);
    void addieren(const Matrix_3x3 & ref_m1, const Matrix_3x3 & ref_m2);
    int determinante() const;
    Matrix_3x3 operator+ (const Matrix_3x3 & ref_m) const;
    Matrix_3x3& operator= (const Matrix_3x3& ref_m);
    bool operator== (const Matrix_3x3& ref_m) const;
    bool operator< (const Matrix_3x3& ref_m) const;
    bool operator> (const Matrix_3x3& ref_m) const;
};

// Konstruktor: 3x3-Matrix initialisieren
Matrix_3x3 :: Matrix_3x3()
{
    for(int i=0;i<3;i++)
        for(int j=0;j<3;j++)
            m[i][j] = 0;
}

// Ausgabe der Matrixelemente
void Matrix_3x3 :: ausgabe() const
{
    cout << endl;
    cout << "Ausgabe" << endl;
    cout << "-----" << endl << endl;
    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++)
            cout << setw(5) << m[i][j];
        cout << endl << endl;
    }
}

// zeilenweises Einlesen der Matrixelemente
void Matrix_3x3 :: einlesen()
{
    cout << endl << endl;
    cout << "Einlesen der Matrix" << endl;
    cout << "-----" << endl << endl;
    for(int i=0;i<3;i++){
        cout << "\nGeben Sie die " << i+1 << ". Zeile ein: ";
        for(int j=0;j<3;j++)
            cin >> m[i][j];
    }
}

```

```

// Setzen eines Wertes an Zeile i und Spalte j
void Matrix_3x3 :: set(int value, int i, int j)
{
    m[i][j] = value;
}

// Ermitteln eines Wertes an Zeile i und Spalte j
int Matrix_3x3 :: get(int i, int j) const
{
    return m[i][j];
}

// Addition der Matrix ref_m zu aktuellen Matrix
void Matrix_3x3 :: addieren(const Matrix_3x3 & ref_m)
{
    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            set(ref_m.get(i,j)+get(i,j),i,j);
        }
    }
}

// Addition der Matrizen ref_m1 und ref_m2 sowie Zuweisung zur aktuellen
void Matrix_3x3 :: addieren(const Matrix_3x3 & ref_m1,
                           const Matrix_3x3 & ref_m2)
{
    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            set(ref_m1.get(i,j)+ref_m2.get(i,j),i,j);
        }
    }
}

// Determinante der Matrix errechnen
int Matrix_3x3 :: determinante() const
{
    return m[0][0]*m[1][1]*m[2][2] +
           m[0][1]*m[1][2]*m[2][0] +
           m[0][2]*m[1][0]*m[2][1] -
           m[0][2]*m[1][1]*m[2][0] -
           m[0][0]*m[1][2]*m[2][1] -
           m[0][1]*m[1][0]*m[2][2];
}

// ueberladener Gleichheits-Operator fuer 3x3-Matrizen
bool Matrix_3x3 :: operator==(const Matrix_3x3& ref_m) const
{
    // paarweise mit aktueller Matrix vergleichen
    for(int i=0; i<3; i++)
        for(int j=0; j<3; j++)
            if(ref_m.get(i,j) != m[i][j])
                return false;
    return true;
}

```

```

// ueberladener Kleiner-Als-Operator fuer 3x3-Matrizen
bool Matrix_3x3 :: operator< (const Matrix_3x3& ref_m) const
{
    // Determinante bestimmt Reihenfolge von Matrizen
    if(ref_m.determinante() > determinante())
        return true;
    else
        return false;
}

// ueberladener Groesser-Als-Operator fuer 3x3-Matrizen
bool Matrix_3x3 :: operator> (const Matrix_3x3& ref_m) const
{
    // Determinante bestimmt Reihenfolge von Matrizen
    if(ref_m.determinante() < determinante())
        return true;
    else
        return false;
}

// ueberladener Kopieroperator fuer 3x3-Matrizen
Matrix_3x3& Matrix_3x3 :: operator= (const Matrix_3x3& ref_m)
{
    // nur Zuweisung, wenn ref_m nicht gleich aktueller Matrix
    if(this != &ref_m){
        // Kopiervorgang starten
        for(int i=0; i<3; i++)
            for(int j=0; j<3; j++)
                set(ref_m.get(i,j),i,j);
    }
    return *this;
}

// ueberladener Additionsoperator fuer 3x3-Matrizen
Matrix_3x3 Matrix_3x3 :: operator+ (const Matrix_3x3 & ref_m) const
{
    // temporaeres Matrix-Objekt erzeugen (mit 0 initialisiert)
    Matrix_3x3 tmp;
    // aktuelle Matrix in temporaere Matrix kopieren
    // oder aktuelle Matrix auf temp. aufaddieren
    tmp.addieren(*this);
    // uebergebene Matrix auf temporaere addieren
    tmp.addieren(ref_m);
    // temporaere Matrix zurueckgeben
    return tmp;
}

```

```

// Hauptprogrammfunktion
int main()
{
    // Matrizenobjekte definieren
    Matrix_3x3 matrix1, matrix2, matrix3, matrix4;

    // matrix1 und matrix2 einlesen
    matrix1.einlesen();
    matrix2.einlesen();
    // matrix1 = matrix1 + matrix2
    matrix1.addieren(matrix2);
    // matrix1 ausgeben
    matrix1.ausgabe();
    // Determinante ausgeben
    cout << "Determinante = " << matrix1.determinante() << endl;

    // matrix3 = matrix1 + matrix2
    matrix3.addieren(matrix1,matrix2);
    // matrix3 ausgeben
    matrix3.ausgabe();
    // Determinante ausgeben
    cout << "Determinante = " << matrix3.determinante() << endl;

    // matrix4.operator=(matrix1.operator+(matrix3)) oder
    // matrix4.addieren(matrix1,matrix4) oder
    matrix4 = matrix1 + matrix3;
    // matrix4 ausgeben
    matrix4.ausgabe();
    // Determinante ausgeben
    cout << "Determinante = " << matrix4.determinante() << endl;

    // matrix4 und matrix3 vergleichen
    if(matrix4 == matrix3)
        cout << "Matrix 4 ist gleich Matrix 3" << endl;
    else{
        if(matrix4 > matrix3)
            cout << "Matrix 4 ist groesser als Matrix 3" << endl;
        else
            cout << "Matrix 4 ist kleiner als Matrix 3" << endl;
    }
    cout << matrix4 << endl;

    return 0;
}

```