# Matrix Operator Overloading

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

class Matrix {
   int mat[3][3];

public:
   // Overloading + operator
   Matrix operator+(Matrix& m);

   // Overloading << operator
   friend ostream& operator<<(ostream& out, Matrix& m);

   // Overloading >> operator
   friend istream& operator>>(istream& in, Matrix& m);
};
Matrix Matrix::operator+(Matrix&m){
   Matrix result;
   for (int i = 0; i < 3; i++) {
       for (int j = 0; j < 3; j++) {
           result.mat[i][j] = this->mat[i][j] + m.mat[i][j];
       }
   }
   return result;
}
ostream& operator<<(ostream& out, Matrix& m) {
   for (int i = 0; i < 3; i++) {
       for (int j = 0; j < 3; j++) {
           out << m.mat[i][j] << " ";
       }
       out << endl;
   }
   return out;
}
istream& operator>>(istream& in, Matrix& m) {
   cout << "Enter elements: ";
   for (int i = 0; i < 3; i++) {
```

```
        for (int j = 0; j < 3; j++) {
            in >> m.mat[i][j];
        }
    }
    return in;
}

int main() {
    Matrix m1, m2, m3;

    cout << "For matrix 1" << endl;
    cin >> m1;
    cout << "For matrix 2" << endl;
    cin >> m2;

    m3 = m1 + m2;

    cout << "Resultant Matrix: " << endl;
    cout << m3;

    return 0;
}
```

**Other Way**

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

class Matrix {
    int** mat;
    int rows, cols;

public:
    // Constructor
    Matrix(int r, int c) : rows(r), cols(c) {
        mat = new int* [rows];
        for (int i = 0; i < rows; ++i)
            mat[i] = new int[cols];
```

```
    }

    // Destructor
    ~Matrix() {
        for (int i = 0; i < rows; ++i)
            delete[] mat[i];
        delete[] mat;
    }

    // Overloading + operator
    Matrix operator+(Matrix& m);

    // Overloading << operator
    friend ostream& operator<<(ostream& out, Matrix& m);

    // Overloading >> operator
    friend istream& operator>>(istream& in, Matrix& m);
};

Matrix Matrix::operator+(Matrix& m) {
    Matrix result(rows, cols);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result.mat[i][j] = this->mat[i][j] + m.mat[i][j];
        }
    }
    return result;
}

ostream& operator<<(ostream& out, Matrix& m) {
    for (int i = 0; i < m.rows; i++) {
        for (int j = 0; j < m.cols; j++) {
            out << m.mat[i][j] << " ";
        }
        out << endl;
    }
    return out;
}

istream& operator>>(istream& in, Matrix& m) {
    cout << "Enter elements: ";
    for (int i = 0; i < m.rows; i++) {
        for (int j = 0; j < m.cols; j++) {
            in >> m.mat[i][j];
        }
```

```
    }
    return in;
}

int main() {
    int rows, cols;
    cout << "Enter number of rows and columns: ";
    cin >> rows >> cols;

    Matrix m1(rows, cols), m2(rows, cols);

    cout << "For matrix 1" << endl;
    cin >> m1;
    cout << "For matrix 2" << endl;
    cin >> m2;

    // Create and assign m3 after m1 and m2 have been filled
    Matrix m3 = m1 + m2;

    cout << "Resultant Matrix: " << endl;
    cout << m3;

    return 0;
}
```

- **Output**

```
Enter number of rows and columns: 2 2
For matrix 1
Enter elements: 4 5 2 1
For matrix 2
Enter elements: 4 5 2 1
Resultant Matrix:
8 10
4 2

D:\MatrixOpera\x64\Debug\MatrixOpera.exe (process
Press any key to close this window . . .
```

- **Matrix Multiplication**

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

class Matrix {
    int** mat;
    int rows, cols;

public:
    // Constructor
    Matrix(int r, int c) : rows(r), cols(c) {
        mat = new int* [rows];
        for (int i = 0; i < rows; ++i)
            mat[i] = new int[cols];
    }

    // Destructor
    ~Matrix() {
        for (int i = 0; i < rows; ++i)
            delete[] mat[i];
        delete[] mat;
    }

    // Overloading * operator
    Matrix operator*(Matrix& m);

    // Overloading << operator
    friend ostream& operator<<(ostream& out, Matrix& m);

    // Overloading >> operator
    friend istream& operator>>(istream& in, Matrix& m);
};

Matrix Matrix::operator*(Matrix& m) {
    if (cols != m.rows) {
        cout << "Matrix multiplication not possible." << endl;
        exit(0);
    }
    Matrix result(rows, m.cols);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < m.cols; j++) {
            result.mat[i][j] = 0;
            for (int k = 0; k < cols; k++) {
                result.mat[i][j] += this->mat[i][k] * m.mat[k][j];
```

```
            }
        }
    }
    return result;
}

ostream& operator<<(ostream& out, Matrix& m) {
    for (int i = 0; i < m.rows; i++) {
        for (int j = 0; j < m.cols; j++) {
            out << m.mat[i][j] << " ";
        }
        out << endl;
    }
    return out;
}

istream& operator>>(istream& in, Matrix& m) {
    cout << "Enter elements: ";
    for (int i = 0; i < m.rows; i++) {
        for (int j = 0; j < m.cols; j++) {
            in >> m.mat[i][j];
        }
    }
    return in;
}

int main() {
    int rows1, cols1, rows2, cols2;
    cout << "Enter number of rows and columns for first matrix: ";
    cin >> rows1 >> cols1;
    cout << "Enter number of rows and columns for second matrix: ";
    cin >> rows2 >> cols2;

    if (cols1 != rows2) {
        cout << "Matrix multiplication not possible." << endl;
        return 0;
    }

    Matrix m1(rows1, cols1), m2(rows2, cols2);

    cout << "For matrix 1" << endl;
    cin >> m1;
    cout << "For matrix 2" << endl;
    cin >> m2;
```

```
    // Create and assign m3 after m1 and m2 have been filled
    Matrix m3 = m1 * m2;

    cout << "Resultant Matrix: " << endl;
    cout << m3;

    return 0;
}
```

- **Output**

```
Microsoft Visual Studio Debu

Enter number of rows and columns for first matrix: 2 2
Enter number of rows and columns for second matrix: 2
2
For matrix 1
Enter elements: 4 5 4 5
For matrix 2
Enter elements: 4 5 2 1
Resultant Matrix:
26 25
26 25

D:\MatrixOpera\x64\Debug\MatrixOpera.exe (process 17612) exited with code 0.
Press any key to close this window . . .
```