```cpp
#include <iostream>
#include <stdexcept>
#include <vector>
using namespace std;

class matrix {
private:
  int _nr;    /* number of rows */
  int _nc;    /* number of columns */
  vector<double> data; /* underlying data storage */
public:
  matrix() : _nr(0), _nc(0), data() {};
  matrix(int numRows, int numColumns, double value=0)
  : _nr(numRows), _nc(numColumns), data(numRows*numColumns, value) {}
  int numRows() const { return _nr; }
  int numColumns() const { return _nc; }
  matrix size() const {
    matrix result(1,2);
    result(0,0) = numRows();
    result(0,1) = numColumns();
    return result;
  }
  bool operator==(const matrix &other) const { return (_nr == other._nr && _nc == other._nc && data == other.data); }
  bool operator!=(const matrix &other) const { return !(this == other); }
};

// provides read-only access to a matrix entry
double& operator[](int r, int c) const {
  if (r < 0 || r >= _nr || c < 0 || c >= _nc)
    throw out_of_range("Invalid indices for matrix");
  return data[r + c * _nr]; // column-major
}

// provides write-access to a matrix entry
double& operator[](int r, int c) {  
  if (r < 0 || r >= _nr || c < 0 || c >= _nc)
     throw out_of_range("Invalid indices for matrix");
  return data[r + c * _nr]; // column-major
}
```
// addition
matrix operator+(const matrix& other) const
{ // produce sum of two matrices
  if (_nr != other._nr || _nc != other._nc)
    throw invalid_argument("Matrix dimensions must agree.");

  matrix result = matrix(*this);
  for (int r=0; r < _nr; r++)
    for (int c=0; c < _nc; c++)
      result(r,c) += other(r,c);
  return result;
}

matrix operator+(double scalar) const
{ // add scalar to all elements
  matrix result = matrix(*this);
  for (int r=0; r < _nr; r++)
    for (int c=0; c < _nc; c++)
      result(r,c) += scalar;
  return result;
}

// multiplication
matrix operator*(double scalar) const
{ // multiply each element by scalar
  matrix result = matrix(*this);
  for (int r=0; r < _nr; r++)
    for (int c=0; c < _nc; c++)
      result(r,c) *= scalar;
  return result;
}

matrix operator*(const matrix& other) const
{ // matrix multiplication
  if (_nc != other._nr)
    throw invalid_argument("Inner matrix dimensions must agree.");

  // *** rest of implementation missing ***
}

// define additional support for reading/writing matrices
ostream& operator<< (ostream& out, const matrix& m);
istream& operator>> (istream& in, matrix& m);