Operator Overloading Workshop Code

Matrix.h

```
#ifndef __MATRIX_H_
#define __MATRIX_H_

class Matrix
{
 private:
   int m_numberOfRows;
   int m_numberOfColumns;
   double *m_matrix;

 public:
   Matrix(){};
   Matrix(int rows, int cols);
  ~Matrix(); //destructor
   void print();//print the content of the matrix
   double&  operator() (int m, int n);              //overloading () opra
   Matrix&  operator = (const Matrix &i_matrix); //oveloading the assi
   Matrix   operator + (const Matrix &i_matrix); //overloading + oprat
   Matrix   operator - (const Matrix &i_matrix); //overloading - opera
   Matrix   operator * (const Matrix &i_matrix); //oveloading  * op2ra
 };
#endif
```

Matrix.cpp

```
Matrix::Matrix(int i_rows, int i_cols)
{
 m_numberOfRows= i_rows ;
 m_numberOfColumns= i_cols;
 m_matrix = new double[m_numberOfRows * m_numberOfColumns];
 for(int i=0; i<m_numberOfRows*m_numberOfColumns; i++)
    m_matrix[i] = (int)(6.0*rand()/(RAND_MAX));
}

Matrix::~Matrix()
{
 delete [] m_matrix;
}

//print the matrix
void Matrix::print()
{
  for(int i=0; i<m_numberOfRows; i++) {
  for(int j=0; j<m_numberOfColumns; j++)
    std::cout<<m_matrix[m_numberOfColumns*i + j]<<"␣";
    std::cout<<std::endl;
  }
   std::cout<<std::endl;
}
```

Matrix.cpp

```
double& Matrix::operator() (int m, int n)
 {
     return m_matrix[m_numberOfColumns*m + n];
 }

//overloading the = operator
Matrix& Matrix::operator = (const Matrix &i_matrix)
{
 for(int i=0; i<m_numberOfRows*m_numberOfColumns; i++)
    m_matrix[i] = i_matrix.m_matrix[i];
 return *this;
}

//overloading the + operator
Matrix Matrix::operator + (const Matrix &i_matrix)
{
 Matrix Mat(m_numberOfRows,m_numberOfColumns);
 for(int i=0; i<m_numberOfRows; i++)
 for(int j=0; j<m_numberOfColumns; j++)
  Mat(i,j) =m_matrix[i*m_numberOfRows + j]
              + i_matrix.m_matrix[i*m_numberOfRows + j];
 return Mat;
}
```

Matrix.cpp

```
//overloading the - operator
Matrix Matrix::operator - (const Matrix &i_matrix)
{
 Matrix Mat(m_numberOfRows,m_numberOfColumns);
 for(int i=0; i<m_numberOfRows; i++)
 for(int j=0; j<m_numberOfColumns; j++)
  Mat(i,j) =m_matrix[i*m_numberOfRows + j] - i_matrix.m_matrix[i*m_nu
 return Mat;
}

//overloading the * operator
Matrix Matrix::operator * (const Matrix &i_matrix)
{
 Matrix Mat(m_numberOfRows,m_numberOfColumns);
 for(int i=0; i<m_numberOfRows; i++)
 for(int j=0; j<m_numberOfColumns; j++)
 {
   double temp=0;
   for(int k=0; k<m_numberOfColumns; k++)
     temp = temp + m_matrix[i*m_numberOfRows + k] * i_matrix.m_matrix
   Mat(i,j)=temp;
 }
 return Mat;
}
```