OPERATOR OVERLOADING !!

#include <iostream>

using namespace std;

template <class M> class Matrix

{

//friend void operator<<(ostream &, const Matrix<M>&);

int NRows, NCols;

M \*arr;

public:

Matrix();

Matrix(int NRows,int NCols);

Matrix(const Matrix<M>&);

~Matrix() {delete [] arr;}

void input();

void display();

int rows() const {return NRows;}

int columns() const {return NCols;}

M& operator()(int,int) const;

Matrix<M>& operator=(const Matrix<M>&);

//Matrix<M> operator+() const;

Matrix <M> operator+(const Matrix<M>&) const;

Matrix<M> operator-() const;

Matrix<M> operator-(const Matrix<M>&) const;

Matrix<M> operator\*(const Matrix<M>&) const;

Matrix<M>& operator+=(const M&);

};

template <class M> Matrix<M>::Matrix()

{

//default constructor

cout<<"Enter the no of rows - ";

cin>>NRows;

cout<<"Enter the no of columns - ";

cin>>NCols;

if(NRows<=0 || NCols<=0)

cout<<"\nError!!! Can't create matrix!! Illegal Row or Column size!!!";

else

arr=new M[NRows\*NCols];

}

template <class M> Matrix<M>::Matrix(int NRows,int NCols)

{

//parameterized constructor

if(NRows<=0 || NCols<=0)

cout<<"\nError!!! Can't create matrix!! Illegal Row or Column size!!!";

else

{

this->NRows=NRows;

this->NCols=NCols;

arr=new M[NRows\*NCols];

}

}

template <class M> void Matrix<M>::input()

{

//taking input for the elements of the array

cout<<"\nEnter the elements -- \n";

for(int i=0;i<NRows;i++)

{

cout<<"\t\t";

for(int j=0;j<NCols;j++)

{

cin>>arr[i\*NCols+j];

}

}

}

template <class M> void Matrix<M>::display()

{

//displaying the array

cout<<"\n\n\t\t";

for(int i=0;i<NRows;i++)

{

for(int j=0;j<NCols;j++)

{

cout<<arr[i\*NCols+j]<<" ";

}

cout<<"\n\t\t";

}

cout<<endl;

}

template <class M> Matrix<M>::Matrix(const Matrix<M>& m)

{

//copy constructor

NRows=m.NRows;

NCols=m.NCols;

arr=new M[NRows\*NCols];

for(int i=0;i<(NRows\*NCols);i++)

{

arr[i]=m.arr[i];

}

}

template <class M> Matrix<M>& Matrix<M>::operator=(const Matrix<M>& m)

{

//overloading assignment operator

if(this!=&m)

{

delete [] arr;

NRows=m.NRows;

NCols=m.NCols;

arr=new M[NRows\*NCols];

for(int i=0;i<(NRows\*NCols);i++)

{

arr[i]=m.arr[i];

}

}

return \*this;

}

template <class M> M& Matrix<M>::operator()(int i,int j) const

{

//overloading ()

if(i<1 || i>NRows || j<1 || j>NCols)

cout<<"\nError!!! Index out of bound!!!";

else

return arr[(i-1)\*NCols+(j-1)];

}

template <class M> Matrix<M> Matrix<M>::operator+(const Matrix<M>& m) const

{

//adding two matrices

if(NRows!=m.NRows || NCols!=m.NCols)

cout<<"\nError!! Matrices size mismatch!!! CANNOT BE ADDED!!";

else

{

Matrix<M> temp(NRows,NCols);

for(int i=0;i<(NRows\*NCols);i++)

temp.arr[i]=arr[i]+m.arr[i];

return temp;

}

}

template <class M> Matrix<M> Matrix<M>::operator\*(const Matrix<M>& m) const

{

//multiplying two matrices, m3=temp\*m2

if(NCols!=m.NRows)

cout<<"\nError!! Matrices size mismatch!!! CANNOT BE MULTIPLIED!!";

else

{

Matrix<M> temp(NRows,m.NCols);

int r=0, curr=0; //r=row\_no(temp); curr=index(m3)

for(int i=0;r<NRows;i+=NCols)

{

r=i; //r+=NCols (starting point of next row)

for(int j=0,c=0;j<m.NCols;j++) //c=col\_no(m2)

{

c=j; //c(starting point of next col)

int sum=0;

for(int k=0;k<m.NRows;k++)

{

sum+=arr[r]\*m.arr[c];

//for iterating in the current col and row

c+=m.NCols,r++;

}

temp.arr[curr++]=sum;

r=i; //going back to the starting of the row

}

}

return temp;

}

}

template <class M> Matrix<M> Matrix<M>::operator-() const

{

//m1=-m2

Matrix<M> temp(NRows,NCols);

for(int i=0;i<(NRows\*NCols);i++)

{

temp.arr[i]=0;

temp.arr[i]-=arr[i];

}

return temp;

}

template <class M> Matrix<M> Matrix<M>::operator-(const Matrix<M>& m) const

{

//subtracting two matrices

if(NRows!=m.NRows || NCols!=m.NCols)

cout<<"\nError!! Matrices size mismatch!!!";

else

{

Matrix<M> temp(NRows,NCols);

for(int i=0;i<(NRows\*NCols);i++)

temp.arr[i]=arr[i]-m.arr[i];

return temp;

}

}

template <class M> Matrix<M>& Matrix<M>::operator+=(const M& m)

{

//m1+=M;

for(int i=0;i<(NRows\*NCols);i++)

arr[i]+=m;

return \*this;

}

/\*template <class M> void operator<<(ostream &out, const Matrix<M>& m)

{

int count=0;

for(int i=0;i<m.NRows;i++)

{

for(int j=0;j<m.NCols;j++)

{

out<<m.arr[count++];

}

out<<endl;

}

return out;

}\*/

int main()

{

Matrix<int> M1;

M1.input();

cout<<"\n\n\n\t\tM1 = ";

M1.display();

cout<<"\nNo of rows in M1 = "<<M1.rows()<<endl;

cout<<"\nNo of columns in M1 = "<<M1.columns()<<endl;

cout<<"\nElement at (1,2) index of M1 = "<<M1(1,2)<<endl;

Matrix<int> M2=M1; //copy constructor

cout<<"\n\n\n\t\tM2 = ";

M2.display();

Matrix<int> M3; //creating a new object

M3.input();

cout<<"\n\n\n\t\tM3 = ";

M3.display();

M2=M3; //assignment operator overload

cout<<"\n\n\n\t\t(UPDATED) M2 = ";

M2.display();

M3=M1+M2; //adding two matrices

cout<<"\n\n\n\t\tM3 = M1+M2";

cout<<"\n\n\n\t\t(UPDATED) M3 = ";

M3.display(); //resultant matrix

Matrix<int> M4;

M4.input();

cout<<"\n\n\n\t\tM4 = ";

M4.display();

Matrix<int> M5=M4\*M2; //multiplying two matrices

cout<<"\n\n\n\t\tM5 = M4\*M2";

cout<<"\n\n\n\t\tM5 = ";

M5.display(); //resultant matrix

M3=M1-M2; //subtracting two matrices

cout<<"\n\n\n\t\tM3 = M1-M2";

cout<<"\n\n\n\t\t(UPDATED) M3 = ";

M3.display(); //resultant matrix

M1+=4; //updating the values

cout<<"\n\n\n\t\t(UPDATED) M1 = ";

M1.display();

return 0;

}