# TASK 1:

**HEADER FILE:**

#include<iostream>

#include<string>

using namespace std;

class Employe

{

public:

string name;

int age;

float salary;

Employe();

friend istream& operator >> (istream& in, Employe obj); // Input

friend Employe& operator + (Employe e1, Employe e2); // Adding two objects

friend Employe& operator - (Employe e1, Employe e2); // Elder Employe

friend Employe& operator == (Employe s1, Employe s2); // Salary Compare

friend ostream& operator << (ostream& out, Employe obj); // Output

};

**FUNCTION FILE:**

#include "Header.h"

Employe :: Employe() {}

istream& operator >> (istream& in, Employe& e)

{

cout << "Enter Name: " << endl;

in >> e.name;

cout << "ENter Age: " << endl;

in >> e.age;

cout << "Enter Salary: " << endl;

in >> e.salary;

return in;

}

Employe& operator + (Employe e1, Employe e2)

{

e1.name = e1.name + e2.name;

e1.age = e1.age + e2.age;

e1.salary = e1.salary + e2.salary;

return e1;

}

Employe& operator - (Employe e1, Employe e2)

{

if (e1.age > e2.age)

{

cout << "Employe 1 is elder" << endl;

return e1;

}

else if (e2.age > e1.age)

{

cout << "Employe 2 is elder" << endl;

return e2;

}

return e1;

}

Employe& operator == (Employe e1, Employe e2)

{

if (e1.salary > e2.salary)

{

cout << "Employe 1 is earning more" << endl;

return e1;

}

else if (e2.salary > e1.salary)

{

cout << "Employe 2 is earning more" << endl;

return e2;

}

}

ostream& operator << (ostream& out, Employe obj)

{

cout << "Name: " << obj.name;

cout << "Age: " << obj.age;

cout << "Salary: " << obj.salary;

return out;

}

**SOURCE FILE:**

#include"Header.h"

int main()

{

Employe e1, e2, e3;

cout << "For Employe 1: " << endl;

cin >> e1;

cout << "For Employe 2: " << endl;

cin >> e2;

e3 = e1 + e2;

e1 - e2;

e1 == e2;

cout << e1;

cout << e2;

}

# Output:

# Task 2:

**HEADER FILE:**

#pragma once

#include<iostream>

using namespace std;

class Matrix

{

int row;

int col;

int\*\* ptr;

public:

Matrix();

Matrix(int row, int col);

friend istream& operator >> (istream& input, Matrix m1);

friend ostream& operator <<(ostream& output, Matrix m1);

Matrix operator +(Matrix m1);

Matrix operator -(Matrix m1);

Matrix operator ++();

Matrix operator ++(int);

Matrix operator --();

Matrix operator --(int);

Matrix operator \*(Matrix& m2);

void disp();

}; **FUNCTION FILE:**

#include"Header.h"

Matrix :: Matrix() :row(0), col(0)

{

ptr = NULL;

}

Matrix :: Matrix(int row, int col)

{

this->row = row;

this->col = col;

ptr = new int\* [row];

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

{

\*(ptr + i) = new int[col];

}

}

istream& operator >> (istream& input, Matrix m1)

{

for (int i = 0; i < m1.row; i++)

{

for (int j = 0; j < m1.col; j++)

{

input >> m1.ptr[i][j];

}

}

return input;

}

ostream& operator <<(ostream& output, Matrix m1)

{

for (int i = 0; i < m1.row; i++)

{

for (int j = 0; j < m1.col; j++)

{

cout << m1.ptr[i][j] << ' ';

}

cout << endl;

}

return output;

}

Matrix Matrix :: operator +(Matrix m1)

{

Matrix temp(row, col);

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

{

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

{

temp.ptr[i][j] = ptr[i][j] + this->ptr[i][j];

}

cout << endl;

}

return temp;

}

Matrix Matrix :: operator -(Matrix m1)

{

Matrix temp(row, col);

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

{

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

{

temp.ptr[i][j] = ptr[i][j] - this->ptr[i][j];

}

cout << endl;

}

return temp;

}

Matrix Matrix :: operator ++()

{

Matrix temp(row, col);

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

{

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

{

++ptr[i][j];

}

cout << endl;

}

return temp;

}

Matrix Matrix :: operator ++(int)

{

Matrix temp(row, col);

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

{

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

{

++ptr[i][j];

}

cout << endl;

}

return temp;

}

Matrix Matrix :: operator --()

{

Matrix temp(row, col);

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

{

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

{

--ptr[i][j];

}

cout << endl;

}

return temp;

}

Matrix Matrix :: operator --(int)

{

Matrix temp(row, col);

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

{

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

{

--ptr[i][j];

}

cout << endl;

}

return temp;

}

Matrix Matrix :: operator \*(Matrix& m2)

{

Matrix temp(row, col);

for (int i = 0; i < row; ++i) {

for (int j = 0; j < col; ++j) {

temp.ptr[i][j] = 0;

}

}

for (int i = 0; i < this->row; ++i)

for (int j = 0; j < m2.col; ++j)

for (int k = 0; k < col; ++k)

{

temp.ptr[i][j] += m2.ptr[i][k] \* ptr[k][j];

}

return temp;

}

void Matrix :: disp()

{

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

{

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

{

cout << ptr[i][j];

}

}

} **SOURCE FILE:**

#include"Header.h"

int main()

{

int row;

int col;

cout << "Enter number of rows :";

cin >> row;

cout << "Enter number of columns :";

cin >> col;

Matrix m1(row, col);

cout << "Enter values of first matrix :" << endl;

cin >> m1;

Matrix m2(row, col);

cout << "Enter values of second matrix :" << endl;

cin >> m2;

Matrix m3;

cout << "Addition of two matrices : " << endl;

m3 = m1 + m2;

cout << m3;

cout << "Subtraction of two matrices is:" << endl;

m3 = m1 - m2;

cout << m3;

cout << "Pre increment operator:" << endl;

++m3;

cout << m3;

cout << "Post increment operator:" << endl;

m3++;

cout << m3;

cout << "Post decrement operator:" << endl;

--m3;

cout << m3;

cout << "Post decrement operator:" << endl;

m3--;

cout << m3;

cout << "Multiplication result is:" << endl;

m3 = m1 \* m2;

cout << m3;

system("pause");

return 0;

}

# OUTPUT:



