Task 1:

MAIN:

#include<iostream>

#include<string>

#include"PersonData.h"

using namespace std;

int main()

{

string str, str2;

int n, n1;

float sal, sal2;

cout << "name : " << endl;

getline(cin, str);

cout << " age : " << endl;

cin >> n;

cout << "salary : " << endl;

cin >> sal;

cin.ignore();

cout << " name : " << endl;

getline(cin, str2);

cout << " age : " << endl;

cin >> n1;

cout << " salary : " << endl;

cin >> sal2;

employee obj(str, n, sal), obj2(str2, n1, sal2);

employee obj3;

obj3 = obj + obj2;

cout << "name after concatenation : " << obj3.Name << endl;

cout << "Age after addition by overloading : " << obj3.Age << endl;

cout << "salary after addition by overloading : " << obj3.Salary << endl;

int temp;

temp = obj.Age > obj2.Age;

if (temp == true)

{

cout << "object 1 age is greater than object 2" << endl << endl;

}

else

{

cout << "object 2 age is greater" << endl << endl;

}

temp = obj.Salary < obj2.Salary;

if (temp == true)

{

cout << "object 1 salary is less than object 2 salary" << endl << endl;

}

else

cout << "object 2 salary is less than object 1 salary" << endl << endl;

system("pause");

return 0;

}

HEADER :

#include<iostream>

#include<string>

using namespace std;

class employee

{

public:

string Name;

int Age;

float Salary;

public:

employee(){

//

}

employee(string name, int age, float salary){

Name = name;

Age = age;

Salary = salary;

}

employee operator +(employee e){

employee temp;

temp.Name = this->Name + e.Name;

temp.Age = this->Age + e.Age;

temp.Salary = this->Salary + e.Salary;

return temp;

}

int operator -(employee e){

if (this->Age > e.Age){

return true;

}

else

return false;

}

bool operator <(employee obj){

if (this->Salary < obj.Salary){

return true;

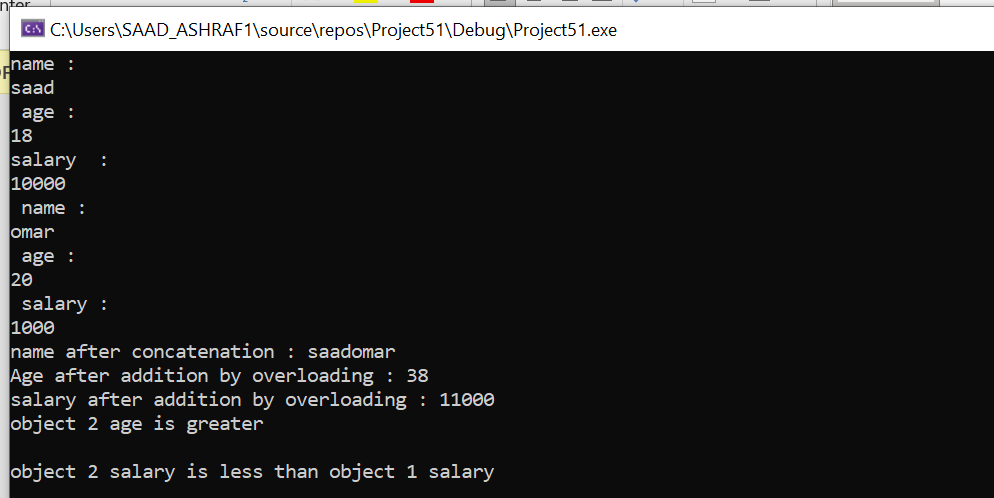
}

else

return false;

}

};



TASK 2:

MAIN:

#include<iostream>

#include"PersonData.h"

using namespace std;

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 MATRICES : " << endl;

m3 = m1 + m2;

cout << m3;

cout << "SUBTRACTION OF MATRICES" << endl;

m3 = m1 - m2;

cout << m3;

cout << "PRE INCREMENT :" << endl;

++m3;

cout << m3;

cout << "POST INCREMENT :" << endl;

m3++;

cout << m3;

cout << "PRE DECREMENT :" << endl;

--m3;

cout << m3;

cout << "POST DECREMENT " << endl;

m3--;

cout << m3;

cout << "MULTIPLICATION :" << endl;

m3 = m1 \* m2;

cout << m3;

system("pause");

return 0;

}

HEADER:

#include<iostream>

using namespace std;

class Matrix {

int row;

int col;

int\*\* ptr;

public:

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

ptr = NULL;

}

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];

}

}

friend istream& operator >> (istream& in, Matrix m) {

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

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

in >> m.ptr[i][j];

}

}

return in;

}

friend ostream& operator <<(ostream& out, Matrix m) {

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

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

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

}

cout << endl;

}

return out;

}

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 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 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 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 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 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 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 disp() {

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

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

cout << ptr[i][j];

}

}

}

};

TASK 3:

MAIN:

#include<iostream>

#include"PersonData.h"

#include<math.h>

using namespace std;

int main() {

float num, num2, num3, num4;

cout << "real number" << endl;

cin >> num;

cout << "imag num" << endl;

cin >> num2;

abc obj(num, num2);

cout << "real number" << endl;

cin >> num3;

cout << "imag num" << endl;

cin >> num4;

abc obj2(num3, num4);

cout << "object 1 increment" << endl;

obj++;

cout << obj;

cout << "object 1 decrement" << endl;

obj--;

cout << obj;

abc obj3;

obj3.add(obj);

cout << "object 3 after addition " << endl;

cout << obj3;

cout << "magnitude" << endl;

cout << obj.mag() << endl;

cout << "multiplication : " << endl;

cout << obj3.mul(obj);

system("pause");

return 0;

}

HEADER :

#include<iostream>

#include<math.h>

using namespace std;

class abc {

float a;

float b;

public:

abc() {}

abc(float x, float y) {

a = x;

b = y;

}

float mag() {

float temp;

temp = sqrt((this->a) \* (this->a) + (this->b) \* (this->b));

return temp;

}

abc add(abc c) {

abc obj;

obj.a = this->a + c.a;

obj.b = this->b + c.b;

return obj;

}

abc operator++() {

abc obj;

++obj.a;

++obj.b;

return obj;

}

abc mul(abc c) {

abc obj;

obj.a = this->a \* c.a;

obj.b = this->b \* c.b;

return obj;

}

abc operator --() {

abc obj;

--obj.a;

--obj.b;

return obj;

}

abc operator --(int) {

abc obj;

--obj.a;

--obj.b;

return obj;

}

abc operator++(int) {

abc obj;

++obj.a;

++obj.b;

return obj;

}

friend ostream& operator<< (ostream& outstream, const abc& obj2) {

outstream << obj2.a << endl;

outstream << obj2.b << endl;

return outstream;

}

};

