TASK#1:

#include<iostream>

#include<iomanip>

#include<string>

using namespace std;

class employ

{

public:

string name;

int age;

float salary;

friend istream& operator >> (istream& in, employ& e);

friend ostream& operator<<(ostream& out, employ& e);

employ operator++()

{

employ;

cout << int(salary / 2) << endl;

return \*this;

}

};

class person

{

public:

string name;

int age;

person operator=(employ& e)

{

cout << "---------------------------------" << endl;

cout << " DATA OF PERSON CLASS" << endl;

cout << "---------------------------------" << endl << endl;

person p;

cout << "Name: ";

p.name = e.name;

cout << p.name << endl;

cout << "Age: ";

p.age = e.age;

cout << p.age << endl;

return p;

}

};

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

{

cout << "Enter name: ";

in >> e.name;

cout << "Enter age: ";

in >> e.age;

cout << "Enter salary: ";

in >> e.salary;

return in;

}

ostream& operator<<(ostream& out, employ& e)

{

out << "Name: " << e.name << endl;

out << "Age: " << e.age << endl;

out << "Salary: " << e.salary << endl;

return out;

}

int main()

{

employ e;

person p;

cout << " INPUT DATA FOR EMPLOY CLASS" << endl;

cout << endl;

cin >> e;

cout << endl;

cout << " OUTPUT DATA OF EMPLOY CLASS" << endl;

cout << endl;

cout << e;

cout << endl;

cout << "---------------------------------" << endl << endl;

cout << "Float salary to int salary: " << endl;

++e;

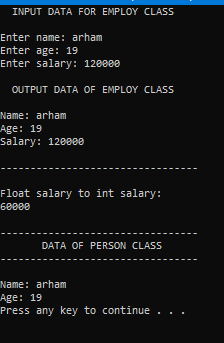
cout << endl;

p = e;

system("pause");

return 0;

}



TASK#2:

#include <iostream>

using namespace std;

class Factorial

{

private:

int num;

public:

Factorial(int f) : num(f) {} // class constructor

double operator!() const // to calculate value of factorial

{

double factorial = 1.0;

for (int i = 1; i <= num; i++)

{

factorial = factorial \* i; // calculating factorial

}

return factorial;

}

};

int main()

{

int num;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl << endl;

cout << " FACTORIAL" << endl;

cout << "-------------------------------------------------------" << endl << endl;

// taking input

cout << "Enter number: ";

cin >> num;

// output

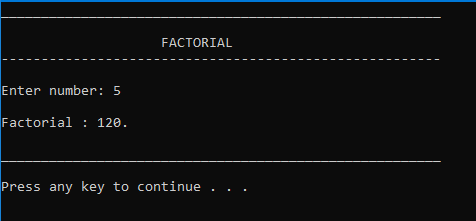
cout << "\nFactorial : " << !Factorial(num) << "." << endl << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl << endl;

system("pause");

return 0;

}



TASK#3:

#include<iostream>

#include<iomanip>

using namespace std;

class matrix

{

public:

int\*\* p, row, col;

matrix(int row, int col)

{

this->row = row;

this->col = col;

p = new int\* [row];

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

{

p[i] = new int[col];

}

}

matrix operator+(const matrix& m) // addition

{

matrix result(3, 3);

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

{

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

{

result.p[i][j] = p[i][j] + m.p[i][j]; //adding matrices

}

}

return result;

}

matrix operator-(const matrix& m) // subtraction

{

matrix result(3, 3);

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

{

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

{

result.p[i][j] = p[i][j] - m.p[i][j]; //subtracting matrices

}

}

return result;

}

void operator++() // for pre-increment

{

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

{

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

{

++p[i][j];

}

}

}

void operator++(int a) // for post increment

{

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

{

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

{

p[i][j]++;

}

}

}

void operator--()// for pre-decrement

{

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

{

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

{

--p[i][j];

}

}

}

void operator--(int a) // for post-decrement

{

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

{

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

{

p[i][j]--;

}

}

}

bool operator<(const matrix& m) // to compare matrix for smaller than relation

{

int x = 0, y = 0;

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

{

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

{

x = x + p[i][j];

y = y + m.p[i][j];

}

}

if (x < y) // condition

return 1;

else

return 0;

}

bool operator>=(const matrix& m) // to compare matrix for greater than or equal relation

{

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

{

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

{

if (p[i][j] < m.p[i][j])

return 0;

}

}

return 1;

}

int operator\*() // multiplication

{

int product = 1;

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

{

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

{

product = product \* p[i][j]; // multiplying matrices

}

}

return product;

}

};

istream& operator>>(istream& input, const matrix& m) // input operation

{

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

{

cout << "Input values for Row " << i + 1 << " :" << endl;

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

{

cout << "Colomn " << j + 1 << " : ";

input >> m.p[i][j];

}

cout << endl;

}

return input;

}

ostream& operator<<(ostream& output, const matrix& m) // output operation

{

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

{

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

{

output << setw(3) << m.p[i][j] << " ";

}

cout << endl;

}

return output;

}

int main()

{

matrix A(3, 3);

matrix B(3, 3);

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl << endl;

cout << " INPUTS" << endl;

cout << "-------------------------------------------------------" << endl << endl;

// taking inputs for 3 x 3 matrices

cout << "Matrix A:" << endl;

cin >> A;

cout << "-------------------------------------------------------" << endl;

cout << "Matrix B:" << endl;

cin >> B;

cout << "-------------------------------------------------------" << endl;

system("pause");

system("cls");

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl << endl;

cout << " OUTPUTS" << endl;

cout << "-------------------------------------------------------" << endl << endl;

// outputs after performing arithematic operation

cout << "Matrices: " << endl << endl;

cout << "A:" << endl;

cout << A << endl;

cout << "B:" << endl;

cout << B << endl;

cout << "-------------------------------------------------------" << endl;

cout << "A + B :" << endl;

cout << A + B;

cout << "-------------------------------------------------------" << endl;

cout << "A - B :" << endl;

cout << A - B;

cout << "-------------------------------------------------------" << endl;

A++;

cout << "Incremented Matrix A:" << endl;

cout << A;

cout << "-------------------------------------------------------" << endl;

A--;

cout << "Decremented Matrix A:" << endl;

cout << A;

cout << "-------------------------------------------------------" << endl;

cout << "Incremented Matrix B:" << endl;

cout << B;

cout << "-------------------------------------------------------" << endl;

A--;

cout << "Decremented Matrix B:" << endl;

cout << B;

cout << "-------------------------------------------------------" << endl;

if (A < B)

{

cout << "A is smaller!" << endl;

}

else

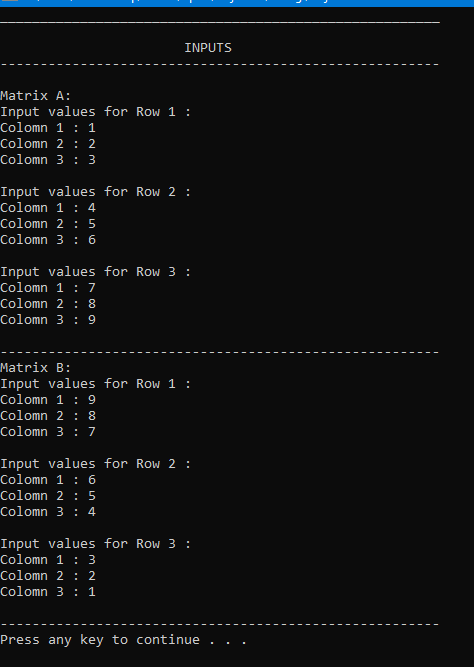
cout << "A is greater or equal to B" << endl << endl;

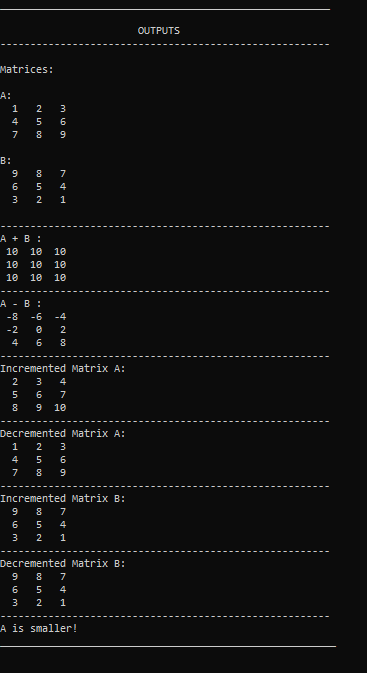
cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

system("pause");

return 0;

}





TASK#4:

#include <iostream>

using namespace std;

class Complex

{

private:

float a;

float b;

public:

Complex() : a(0), b(0)

{ }

void inputnumber() //input

{

cout << "Enter real part of complex number: "; // real part

cin >> a;

cout << "Enter imaginary part of complex number: "; // imaginary part

cin >> b;

}

float mag() // magnitude

{

int mag;

mag = sqrt(pow(a, 2) + pow(b, 2));

return mag;

}

Complex add(Complex c) //sum

{

Complex result;

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

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

return result;

}

Complex mul(Complex c) // product

{

Complex mul;

mul.a = (this->a) \* c.a - (this->b) \* c.b;

mul.b = (this->a) \* c.b + (this->b) \* c.a;

return mul;

}

Complex operator++() //pre-increment

{

++a;

++b;

return \*this;

}

Complex operator++(int x) // post-increment

{

a++;

b++;

return \*this;

}

Complex operator--() // pre-decrement

{

--a;

--b;

return \*this;

}

Complex operator--(int x) // post-decrement

{

a--;

b--;

return \*this;

}

friend ostream& operator<<(ostream& out, Complex number); // friend fuction for output

};

ostream& operator<<(ostream& out, Complex number) // output

{

out << number.a << "+" << number.b << "i" << endl; // complex number format

return out;

}

int main()

{

Complex c1, c2;

// input numbers

cout << "------------------------------------" << endl;

cout << "1st COMPLEX NUMBER" << endl;

cout << "------------------------------------" << endl << endl;

c1.inputnumber();

cout << "------------------------------------" << endl;

cout << "2nd COMPLEX NUMBER" << endl;

cout << "------------------------------------" << endl << endl;

c2.inputnumber();

cout << "------------------------------------" << endl << endl;

system("pause");

system("cls");

// display numbers

cout << "\n1st COMPLEX NUMBER: " << c1 << endl;

cout << "2nd COMPLEX NUMBER: " << c2 << endl;

cout << "------------------------------------" << endl << endl;

c1.add(c2);

cout << "SUM:\n" << c1.add(c2);

c1.mul(c2);

cout << "\nPRODUCT:\n" << c1.mul(c2);

cout << "------------------------------------" << endl;

cout << "1st COMPLEX NUMBER" << endl;

cout << "------------------------------------" << endl << endl;

cout << "MAGNITUDE: "; // magnitude

cout << c1.mag() << endl;

cout << "PRE-INCREMENT: ";

++c1;// pre-increment

cout << c1;

cout << "POST-INCREMENT: ";

c1++;// post-increment

cout << c1;

cout << "PRE-DECREMENT: ";

--c1;// pre-decremnt

cout << c1;

cout << "POST-DECREMENT: ";

c1--;// post-decremnt

cout << c1;

cout << "\n------------------------------------" << endl;

cout << "2nd COMPLEX NUMBER" << endl;

cout << "------------------------------------" << endl << endl;

cout << "MAGNITUDE: "; // magnitude

cout << c2.mag() << endl;

cout << "PRE-INCREMENT: ";

++c2;// pre-increment

cout << c2;

cout << "POST-INCREMENT: ";

c2++;// post-increment

cout << c2;

cout << "PRE-DECREMENT: ";

--c2;// pre-decremnt

cout << c2;

cout << "POST-DECREMENT: ";

c2--;// post-decremnt

cout << c2;

system("pause");

return 0;

}

