Nicoleta.Radu.T4

Programare OOP – c++

# Polinom

#include <iostream>

#include <vector>

using namespace std;

class Polinom

{

private:

int grad;

vector <float> coeficient;

public:

Polinom(int);

~Polinom()

{

}

void citire();

void afisare();

void dealocare();

Polinom operator+(Polinom);

float calculPunct(float);

};

inline Polinom::Polinom(int g)

{

grad = g;

//[x.grad + 1];

for (int i = 0; i < grad + 1; i++)

{

coeficient.push\_back(0);

}

}

void Polinom::dealocare()

{

grad = 0;

coeficient.clear();

}

void Polinom::citire()

{

cout << "Introduceti coeficientii polinomului" << endl;

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

{

cout << "[" << i << "]= ";

cin >> coeficient.at(i);

}

}

void Polinom::afisare()

{

cout << coeficient.at(grad) << " x^" << grad;

for (int i = grad - 1; i >= 1 ; i--)

{

if (coeficient.at(i) > 0)

{

cout << " + " << coeficient.at(i) << " \* x^" << i << " ";

}

else

{

if (coeficient.at(i) < 0)

{

cout << coeficient.at(i) << " \* x^" << i << " ";

}

}

}

if (coeficient.at(0) > 0)

{

cout << " + " << coeficient.at(0) << " ";

}

else

{

if (coeficient.at(0) < 0)

{

cout << coeficient.at(0) << " ";

}

}

cout << endl;

}

Polinom Polinom::operator+(Polinom x)

{

// cazul 1 - Gradele sunt egale

if (grad == x.grad)

{

Polinom rezultat(grad);

for (int i = 0; i < rezultat.grad; i++)

{

rezultat.coeficient.at(i) = coeficient.at(i) + x.coeficient.at(i);

}

return rezultat;

}

// cazul 2 - Gradul polinom. 1 este mai mare decat polinom 2

if (grad > x.grad)

{

Polinom rezultat(grad);

for (int i = 0; i < x.grad; i++)

{

rezultat.coeficient[i] = coeficient[i] + x.coeficient[i];

}

for (int i = x.grad + 1; i <= rezultat.grad; i++)

{

rezultat.coeficient[i] = coeficient[i];

}

return rezultat;

}

// cazul 3 - Gradul polinom. 2 mai mare decat polinom 1

if (x.grad > grad)

{

Polinom rezultat(x.grad);

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

{

rezultat.coeficient[i] = coeficient[i] + x.coeficient[i];

}

for (int i = grad + 1; i <= rezultat.grad; i++)

{

rezultat.coeficient[i] = x.coeficient[i];

}

return rezultat;

}

}

float Polinom::calculPunct(float x)

{

float s = 0;

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

{

s = s + coeficient[i] \* pow(x, (float)i);

}

return s;

}

APEL

int n;

cout << "Dati gradul polinoamelor: ";

cin >> n;

Polinom a(n);

Polinom b(n);

a.citire();

b.citire();

cout << endl;

a.afisare();

b.afisare();

Polinom rez(4);

rez = a + b;

cout << "Suma dintre a si b" << endl;

rez.afisare();

cout << "Valoarea polinomului intr-un punct" << endl;

cout << "Dati punctul: ";

float x{};

cin >> x;

cout << "Valoarea lui a intr-un in " << x << " este " << a.calculPunct(x) << endl;

cout << endl;

# Stiva

#include <iostream>

using namespace std;

class Stiva

{

private:

int dim{0};

int limit{0};

int stiv[30];

public:

Stiva();

int isEmpty();

int isFull();

void push(int);

void pop();

void write();

};

inline Stiva::Stiva(): limit(sizeof(stiv) / sizeof(stiv[0]))

{

}

int Stiva::isEmpty()

{

if (dim == 0)

{

return 1;

}

return 0;

}

int Stiva::isFull()

{

if (isEmpty() == 1)

{

return 0;

}

return 1;

}

void Stiva::push(int e)

{

if (isFull() == 1)

{

cout << "Stiva este plina";

}

stiv[dim++] = e;

}

void Stiva::pop()

{

if (isEmpty() == 1)

{

cout << "Stiva este goala";

}

stiv[dim--];

}

void Stiva::write()

{

cout << "stiva: " << endl;

for (size\_t i = 0; i < dim; i++)

{

cout << stiv[i] << " ";

}

}

APEL

Stiva test;

test.isEmpty();

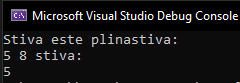
test.push(5);

test.push(8);

test.write();

test.pop();

test.write();



# Numere rationale

#include<iostream>

using namespace std;

class Fractie

{

private:

// partile unei fractii, numitor si numarator

double numitor, numarator;

public:

void citire();

void scriere();

Fractie operator+(Fractie);

Fractie operator-(Fractie);

Fractie operator\*(Fractie);

Fractie operator/(Fractie);

Fractie operator^(int);

private:

Fractie inversa();

};

void Fractie::citire()

{

cout << "Numitor: ";

cin >> numitor;

cout << "Numarator: ";

cin >> numarator;

}

void Fractie::scriere()

{

cout << "Fractia este: " << numarator << " / " << numitor << endl;

}

Fractie Fractie::inversa()

{

Fractie rezultat;

rezultat.numitor = numarator;

rezultat.numarator = numitor;

return rezultat;

}

Fractie Fractie::operator+(Fractie A)

{

Fractie rezultat;

double aux;

if (numitor > A.numitor)

{

aux = numitor / A.numitor;

rezultat.numitor = A.numitor \* aux;

}

else if (numitor < A.numitor)

{

aux = A.numitor / numitor;

rezultat.numitor = numitor \* aux;

}

rezultat.numarator = numarator + A.numarator;

return rezultat;

}

Fractie Fractie::operator-(Fractie A)

{

Fractie rezultat;

double aux;

if (numitor > A.numitor)

{

aux = numitor / A.numitor;

rezultat.numitor = A.numitor \* aux;

}

else if (numitor < A.numitor)

{

aux = A.numitor / numitor;

rezultat.numitor = numitor \* aux;

}

rezultat.numarator = numarator - A.numarator;

return rezultat;

}

Fractie Fractie::operator\*(Fractie A)

{

Fractie rezultat;

rezultat.numarator = numarator \* A.numarator;

rezultat.numitor = numitor \* A.numitor;

return rezultat;

}

Fractie Fractie::operator/(Fractie A)

{

Fractie rezultat;

rezultat.numarator = numarator;

rezultat.numitor = numitor;

rezultat = rezultat \* A.inversa();

return rezultat;

}

Fractie Fractie::operator^(int A)

{

Fractie rezultat;

rezultat.numitor = pow(numitor, A);

rezultat.numarator = pow(numarator, A);

return rezultat;

}

**APEL**

Fractie a{}, b{}, rezultat{};

a.citire();

b.citire();

a.scriere();

b.scriere();

cout << "Adunare" << endl;

rezultat = a + b;

rezultat.scriere();

cout << "Scadere" << endl;

rezultat = a - b;

rezultat.scriere();

cout << "Inmultire" << endl;

rezultat = a \* b;

rezultat.scriere();

cout << "Impartire" << endl;

rezultat = a / b;

rezultat.scriere();

cout << "Ridicare la putere cu 2" << endl;

rezultat = a^2;

rezultat.scriere();

