Laborator 10

```
/* Lab 10  Popa Larisa-Ancuta  Prob 1

-implementați programul prezentat în exemplul 3 și examinați eventualele erori date la
compilare daca exista prin eliminarea comentariilor
-modificați programul astfel încât să se poată accesa din funcția main( ), prin
intermediul obiectului obiect_derivat,
   și metodele aduna( ) și scade( ) din clasa de baza pastrand mostenirea de tip private

*/

//main
#include <iostream>

using namespace std;

#include "Baza_deriv.h"

int main()
{
	Baza obiect_baza;
	cout << "\nAfis din baza (val. initiale): " << obiect_baza.getA() << " " <<
obiect_baza.getB() << '\n';
	cout << "\nSuma este (cu val. initiale, baza) = " << obiect_baza.aduna(); //
corect aduna( ) e public
	cout << "\n\nDiferenta este (cu val. initiale, baza) = " << obiect_baza.scade();
//corect scade( ) e public

	obiect_baza.setA(2);
	obiect_baza.setB(3);

	cout << "\n\nAfis din baza (modificat): " << obiect_baza.getA() << " " <<
obiect_baza.getB() << '\n';
	cout << "\nSuma/Diferenta dupa setare= " << obiect_baza.aduna() << "/" <<
		obiect_baza.scade() << '\n';

	Derivata obiect_derivat;
	cout << "\nProdusul este (din derivat cu val. initiale) = " <<
obiect_derivat.inmulteste() << '\n';
	cout << "\nSuma este (din derivat cu val. initiale, baza) = " <<
obiect_derivat.aduna( );
	cout << "\n\nDiferenta este (din derivat cu val. initiale, baza) = " <<
obiect_derivat.scade() << endl << endl;
}

//header
#pragma once

class Baza
{
protected:
	int a, b;
public:
	Baza() { a = 1, b = 1; }
```

```
	void setA(int a) { this->a = a; }
	void setB(int b) { this->b = b; }

	int getA() { return a; }
	int getB() { return b; }

	int aduna() { return a + b; }
	int scade() { return a - b; }
};

class Derivata : public Baza
{
public:
	int inmulteste() { return a * b; }

	int aduna() { return Baza::aduna(); }
	int scade() { return Baza::scade(); }
};
```

```
/*  Lab 10  Popa Larisa-Ancuta  Prob 2

-folosind modelul claselor de la mostenirea publica, implementați două clase, astfel:
   -clasa de bază contine metode pentru:
     -codarea unui șir de caractere (printr-un algoritm oarecare) => public;
     -afișarea șirului original și a celui rezultat din transformare = > public;
   -clasa derivata contine o metoda pentru:
     -scrirea rezultatului codării într-un fișier, la sfârșitul acestuia.
-fiecare inregistrare are forma: nr_inregistrare: șir_codat;
-accesul la metodele ambelor clase se face prin intermediul unui obiect rezultat prin
instanțierea clasei derivate
-programul care folosește clasele citește un șir de caractere de la tastatură și apoi, în
funcție de opțiunea utilizatorului,
-afișează rezultatul codării sau îl scrie în fișier
*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <fstream>

using namespace std;

#include "Baza.h"
#include "Derivata.h"

int main()
{
	char sir[dim];
	int optiune, ok;

	do
	{
		Derivata o;

		cout << "\nIntroduceti un sir de caractere: ";
		cin >> sir;
```

```
		o.setSir(sir);

		cout << "\nIntroduceti 0 -> afisare pe ecran / 1 -> afisare in fisier: ";
		cin >> optiune;

		if (optiune == 0)
			o.afisare();
		else
		{
			if (optiune == 1)
				o.scrie_fisier();
			else
				cout << "\nAti introdus o valoare gresita!\n\n";
		}

		cout << "\nDoriti sa continuati? (1-DA) ";
		cin >> ok;

	} while (ok == 1);

	return 0;
}

//header
//clasa Baza
#pragma once

const int dim = 50;

class Baza
{
	char sir[dim];
public:
	Baza()
	{
		strcpy(sir, "-");
	}

	void setSir(char*);

	char* codare();
	void afisare();
};

void Baza::setSir(char *s)
{
	if (s != 0)
		strcpy(sir, s);
	else
		strcpy(sir, "-");
}

char* Baza::codare()
{
	char *aux = new char[dim];
	int i;

	for (i = 0;i < strlen(sir);i++)
```

```
			aux[i] = sir[i] + 1;

	aux[strlen(sir)] = '\0';

	return aux;
}

void Baza::afisare()
{
	cout << "\nSirul initial: " << sir;
	cout << "\nSirul codat: " << codare() << endl;
}

//clasa Derivata
#pragma once

class Derivata : public Baza
{
	static int n;
public:
	void scrie_fisier();
};

void Derivata::scrie_fisier()
{
	n++;

	ofstream fout("Date.out");

	char aux[dim];

	strcpy(aux, codare());

	fout << "\nNr_inregistrare: " << n;
	fout << "\nSir_codat: " << aux << endl;

	fout.close();
}

int Derivata :: n = 0;
```

```
/*  Lab 10  Popa Larisa-Ancuta  Prob 3

-să se implementeze o clasă de bază cu două atribute protected de tip întreg care conține
o metoda mutator pentru fiecare atribut al clasei,
   parametri metodelor fiind preluati in main() de la tastatura și metode accesor pentru
fiecare atribut care returneaza atributul specific
-să se scrie o a doua clasă, derivată din aceasta, care implementează operațiile
matematice elementare: +, -, *, /
   asupra atributelor din clasa de bază, rezultatele fiind returnate de metode
-să se scrie o a III-a clasă, derivată din cea de-a doua, care implementează în plus o
metoda pentru
   extragerea rădăcinii pătrate dintr-un număr ( mul, rezultat al operatiei * din prima
clasa derivata)
   de ridicare la putere (atât baza (plus, rezultat al operatiei + din prima clasa
derivata)
```

```
   puterea (minus, rezultat al operatiei - din prima clasa derivata) sunt trimiși ca
parametri)
-verificati apelul metodelor considerand obiecte la diferite ierarhii

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include "Baza.h"
#include "Derivata.h"
#include "Metode.h"

int main()
{
	Derivata o;
	Metode m;

	int val;

	cout << "\nx=";
	cin >> val;
	o.setX(val);
	cout << "y=";
	cin >> val;
	o.setY(val);

	cout << "\n\nx+y= " << o.adunare() << "\nx-y=" << o.scadere() << "\nx*y=" <<
o.inmultire() << "\nx/y=" << o.impartire();
	cout << "\n\nsqrt(x*y)=" << m.radical() << "\nplus^minus=" <<
m.putere(o.adunare(), o.scadere()) << endl << endl;

	return 0;
}

//header
//clasa Baza
#pragma once

class Baza
{
protected:
	int x, y;
public:
	Baza()
	{
		x = 0;
		y = 1;
	}

	void setX(int val) { x = val; }
	void setY(int val) { y = val; }

	int getX() { return x; }
	int getY() { return y; }
```

```
};
```

```
//clasa Derivata
#pragma once

class Derivata :public Baza
{
protected:
	int plus = adunare(), minus = scadere(), mul = inmultire();
public:
	int adunare() { return x + y; }
	int scadere() { return x - y; }
	int inmultire() { return x*y; }
	float impartire() { return (float)x / y; }
};
```

```
//clasa Metode
#pragma once

class Metode :public Derivata
{
public:
	float radical()//calculat cu valorile implicite
	{
		return sqrt(mul);
	}

	int putere(int plus,int minus)
	{
		return pow(plus, minus);
	}
};
```

```
/* Lab 10  Popa Larisa-Ancuta  Prob 4

-definiţi o clasă numita Triangle care are 3 atribute protected pentru laturi si o metoda
care calculează perimetrul unui triunghi
   ale cărui laturi sunt citite de la tastatura (folosite de un constructor adecvat)
-apoi o clasă derivata in mod public din Triangle, Triangle_extended, care în plus,
calculează şi aria triunghiului
-folosind obiecte din cele doua clase apelati metodele specifice
-verificati inainte de instantiere posibilitatea definirii unui triunghi

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include "Triangle.h"
#include "Triangle_extended.h"

int main()
{
```

```
	int l1, l2, l3;
	cout << "\nIntroduceti laturile unui triunghi: \n";
	cin >> l1 >> l2 >> l3;

	if (l1 + l2 > l3 && l1 + l3 > l2 && l2 + l3 > l1)
	{
		Triangle T(l1, l2, l3);

		cout << "\nPerimetrul triunghiului este: " << T.perimetru();

		Triangle_extended T2(l1, l2, l3);

		cout << "\nAria triunghiului este: " << T2.aria() << endl << endl;
	}
	else
		cout << "\nValorile introduse nu pot forma un triunghi!\n\n";
	return 0;
}
```

```
//header
//clasa Triangle
#pragma once

class Triangle
{
protected:
	int a, b, c;
public:
	Triangle()
	{
		a = 2;
		b = 3;
		c = 4;
	}

	Triangle(int l1,int l2,int l3)
	{
		a = l1;
		b = l2;
		c = l3;
	}

	int perimetru() { return a + b + c; }
};
```

```
//clasa Triangle_extended
#pragma once

class Triangle_extended :public Triangle
{
public:
	Triangle_extended(int l1, int l2, int l3)
	{
		a = l1;
		b = l2;
		c = l3;
	} //se foloseste pentru a lucra cu valorile introduse de la tastatura
```

```
	float aria()
	{
		float p = (float)perimetru() / 2;

		return sqrt(p*(p - a)*(p - b)*(p - c));
	}
};
```

```
/* Lab 10  Popa Larisa-Ancuta  Prob 5

-adaugaţi în clasa derivată din exemplul anterior o metodă care calculează înalţimea
triunghiului
-apelati metoda folosind un obiect adecvat

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include "Triangle.h"
#include "Triangle_extended.h"

int main()
{
	int l1, l2, l3;
	cout << "\nIntroduceti laturile unui triunghi: \n";
	cin >> l1 >> l2 >> l3;

	if (l1 + l2 > l3 && l1 + l3 > l2 && l2 + l3 > l1)
	{
		Triangle T(l1, l2, l3);

		cout << "\nPerimetrul triunghiului este: " << T.perimetru();

		Triangle_extended T2(l1, l2, l3);

		cout << "\nAria triunghiului este: " << T2.aria();
		cout << "\nInaltimea triunghiului este: " << T2.inaltimea() << endl <<
endl;
	}
	else
		cout << "\nValorile introduse nu pot forma un triunghi!\n\n";
	return 0;
}

//header
//clasa triangle
#pragma once

class Triangle
{
protected:
	int a, b, c;
```

```
public:
	Triangle()
	{
		a = 2;
		b = 3;
		c = 4;
	}

	Triangle(int l1, int l2, int l3)
	{
		a = l1;
		b = l2;
		c = l3;
	}

	int perimetru() { return a + b + c; }
};
```

```
//clasa Triangle_extended
#pragma once

class Triangle_extended :public Triangle
{
public:
	Triangle_extended(int l1, int l2, int l3)
	{
		a = l1;
		b = l2;
		c = l3;
	} //se foloseste pentru a lucra cu valorile introduse de la tastatura

	float aria()
	{
		float p = (float)perimetru() / 2;

		return sqrt(p*(p - a)*(p - b)*(p - c));
	}

	float inaltimea()
	{
		return (aria() * 2) / a; //a-latura pe care cade inaltimea
	}
};
```

```
/*  Lab 10  Popa Larisa-Ancuta  Prob 6

-definiți o clasă numită Forme care defineste o figura geometrica cu un nume ca si
atribut de tip pointer la un sir de caractere
-clasa va contine un constructor fara parametrii, unul cu parametrii, copy constructor si
se va supraincarca operatorul de asignare
-clasa va avea si o metoda getter si un destructor
-derivati in mod public o clasa Cerc care adauga un atribut de tip int pentru raza si
constructori adecvati considerand si atributele
   (nume, raza) si o metoda getter pentru raza si alte metode care calculeaza aria si
perimetrul cercului de raza r, valoare introdusa
   in main( ) de la tastatura
```

```
-similar definiti o clasa Patrat si Dreptunghi care permit determinarea ariei si
perimetrului obiectelor specifice
-instantiati obiecte din clasele derivate si afisati aria si perimetrul obiectelor
-datele specifice vor fi introduse de la tastatura
-definiti un obiect de tip Cerc cu parametrii care sa il copiati intr-un nou obiect la
care sa ii afisati atributele
-definiti un obiect de tip Patrat cu parametrii si altul fara parametrii
-asignati celui fara parametrii obiectul instantiat cu parametrii si afisati atributele

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include "Forme.h"
#include "Cerc.h"
#include "Patrat.h"
#include "Dreptunghi.h"

#define DIM 50

int main()
{
	char n[DIM];
	int val;

	Cerc c;

	cout << "\nIntroduceti numele cercului: ";
	cin >> n;
	c.setNume(n);
	cout << "\nIntroduceti raza cercului: ";
	cin >> val;
	c.setRaza(val);

	cout << "\nPerimetrul cercului " << c.getNume() << " este: " << c.perimetru();
	cout << "\nAria cercului " << c.getNume() << " este: " << c.aria() << endl;


	Patrat p;

	cout << "\nIntroduceti numele patratului: ";
	cin >> n;
	p.setNume(n);
	cout << "\nIntroduceti latura patratului: ";
	cin >> val;
	p.setLatura(val);

	cout << "\nPerimetrul patratului " << p.getNume() << " este: " << p.perimetru();
	cout << "\nAria patratului " << p.getNume() << " este: " << p.aria() << endl;


	Dreptunghi d;

	cout << "\nIntroduceti numele dreptunghiului: ";
```

```
	cin >> n;
	d.setNume(n);
	cout << "\nIntroduceti lungimea dreptunghiului: ";
	cin >> val;
	d.setLungime(val);
	cout << "\nIntroduceti latimea dreptunghiului: ";
	cin >> val;
	d.setLatime(val);

	cout << "\nPerimetrul dreptunghiului " << d.getNume() << " este: " <<
d.perimetru();
	cout << "\nAria dreptunghiului " << d.getNume() << " este: " << d.aria() << endl
<< endl;

	/*Cerc C2(3, "cerc");
	Cerc C3 = C2;

	cout << "\nNumele cercului 3 este " << C3.getNume() << " cu raza egala cu: " <<
C3.getRaza();*/

	return 0;
}

//header
//clasa Forme
#pragma once

class Forme
{
public :
	char *nume;

	Forme() //constructor fara parametrii
	{
		nume = new char[strlen("-")];
		strcpy(nume, "-");
	}
	Forme(char *n) //constructor cu parametrii
	{
		if (n != 0)
			strcpy(nume, n);
		else
			strcpy(nume, "-");
	}

	void setNume(char *n);
	Forme(const Forme&); //copy constructor

	char* getNume() { return nume; }

	~Forme() { delete[] nume; } //destructor
};

void Forme::setNume(char *n)
{
	nume = new char[strlen(n)];
	strcpy(nume, n);
}
```

```
Forme::Forme(const Forme& f)
{
	nume = new char[strlen(f.nume) + 1];
	strcpy(nume, f.nume);
}
```

```
//clasa Cerc
#pragma once

const float PI = 3.14;

class Cerc :public Forme
{
	char *nume;
	int r;
public:
	Cerc()
	{
		r = 1;
	}
	Cerc(int val, char *n)
	{
		r = val;
		if (n != 0)
			strcpy(nume, n);
		else
			strcpy(nume, "-");
	}

	void setRaza(int val) { r = val; }
	int getRaza() { return r; }

	float perimetru() { return 2 * PI*r; }
	float aria() { return PI*pow(r, 2); }
};
```

```
//clasa Patrat
#pragma once

class Patrat :public Forme
{
	char *nume;
	int l;
public:
	Patrat()
	{
		l = 1;
	}

	void setLatura(int val) { l = val; }
	int getLatura() { return l; }

	float perimetru() { return 4 * l; }
	float aria() { return pow(l, 2); }
};
```

```
//clasa Dreptunghi
#pragma once

class Dreptunghi :public Forme
{
	char *nume;
	int l, L;
public:
	Dreptunghi()
	{
		l = 1;
		L = 1;
	}

	void setLungime(int val) { L = val; }
	void setLatime(int val) { l = val; }
	int getLungime() { return L; }
	int getLatime() { return l; }

	float perimetru() { return 2 * L + 2 * l; }
	float aria() { return l*L; }
};
```

```
/*  Lab 10  Popa Larisa-Ancuta  Prob 7

-considerați o clasa de baza Cerc definita printr-un atribut protected raza, care are un
constructor cu parametrii
-si o metoda care determina aria cercului
-considerați o alta clasa de baza Patrat cu un atribut protected latura similar clasei
Cerc
-derivați un mod public clasa CercPatrat care are un constructor ce apelează
constructorii claselor de baza
-si o metoda care verifica daca pătratul de latura l poate fi inclus in cercul de raza r
-de asemenea clasa derivata determina si perimetrul celor doua figuri geometrice
-instanțiați un obiect din clasa derivata (datele introduse de la tastatura), determinați
aria si perimetrul cercului si al pătratului
-afișați daca pătratul cu latura introdusa poate fi inclus in cercul de raza specificat

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include "Cerc.h"
#include "Patrat.h"
#include "CercPatrat.h"

int main()
{
	int r, l;

	cout << "\nIntroduceti raza cercului: ";
	cin >> r;
```

```
	cout << "\nIntroduceti latura patratului: ";
	cin >> l;
	CercPatrat CP(r, l);

	cout << "\nAria cercului: " << CP.ariaC();
	cout << "\nAria patratului: " << CP.ariaP();

	CP.inclus();

	cout << "\nPerimetrul cercului: " << CP.perimetruC();
	cout << "\nPerimetrul patratului: " << CP.perimetruP() << endl << endl;

	return 0;
}
```

```
//header
//clasa Cerc
#pragma once

const float PI = 3.14;

class Cerc
{
protected:
	int raza;
public:
	Cerc()
	{
		raza = 1;
	}
	Cerc(int val) { raza = val; }

	float ariaC() { return PI*pow(raza, 2); }
};
```

```
//clasa Patrat
#pragma once

class Patrat
{
protected:
	int latura;
public:
	Patrat()
	{
		latura = 1;
	}
	Patrat(int val) { latura = val; }

	int ariaP() { return pow(latura, 2); }
};
```

```
//clasa CercPatrat
#pragma once

class CercPatrat :public Cerc, public Patrat
{
public:
```

```
	CercPatrat(int r, int l) :Cerc(r), Patrat(l)
	{
		raza = r;
		latura = l;
	}

	void inclus()
	{
		if (2 * raza > latura)
			cout << "\n\nPatratul este inclus in cerc.\n";
		else
			cout << "\n\nPatratul nu este inclus in cerc.\n";
	}

	float perimetruC() { return 2 * PI*raza; }
	int perimetruP() { return 4 * latura; }
};
```

```
/*  Lab 10  Popa Larisa-Ancuta  Prob 8

-considerați clasa Fractie care are doua atribute întregi protected a si b pentru
numărător si numitor, doua metode de tip set( )
   respectiv get( ) pentru fiecare din atributele clasei
-declarați o metoda publica simplifica( ) care simplifica un obiect Fractie
-definiti un constructor explicit fara parametri care initializeaza a cu 0 si b cu 1, si
un constructor explicit cu doi parametri care
   va putea fi apelat daca se verifica posibilitatea definirii unei fractii (b!=0)
-supraîncărcați operatorii de adunare, scadere, inmultire si impartire (+,-,*,/) a
fracțiilor folosind metode membre care si simplifica
   daca e cazul rezultatele obtinute, apeland metoda simplifica( ) din clasa
-definiți o clasa Fractie_ext derivata public din Fractie, care va avea un constructor cu
parametrii (ce apelează constructorul din clasa de baza)
-supraîncărcați operatorii de incrementare si decrementare prefixați care aduna/scade
valoarea 1 la un obiect de tip Fractie_ext cu metode membre.
-instanțiați doua obiecte de tip Fractie fără parametrii
-setați atributele obiectelor cu date citite de la tastatura
-afișați atributele inițiale ale obiectelor si noile atribute definite
-efectuați operațiile implementate prin metodele membre, inițializând alte 4 obiecte cu
rezultatele obținute
-simplificați si afișați rezultatele
-instanțiați doua obiecte de tip Fractie_ext cu date citite de la tastatura
-efectuați operațiile disponibile clasei, asignând rezultatele obținute la alte obiecte
Fracti_ext
-simplificați si afișați rezultatele

*/

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

#include "Fractie.h"
#include "Fractie_ext.h"
```

```cpp
int main()
{
	Fractie f1, f2;
	int nr, nm;

	cout << "\nFractia 1: " << f1.getA() << "/" << f1.getB();
	cout << "\nFractia 2: " << f2.getA() << "/" << f2.getB();

	cout << "\n\nIntroduceti valori noi: \nFractia 1:\n  a=";
	cin >> nr;
	cout << "  b=";
	cin >> nm;

	f1.setA(nr);
	f1.setB(nm);

	cout << "\n\nFractia 2:\n  a=";
	cin >> nr;
	cout << "  b=";
	cin >> nm;

	f2.setA(nr);
	f2.setB(nm);

	cout << "\n\nFractia 1: " << f1.getA() << "/" << f1.getB();
	cout << "\nFractia 2: " << f2.getA() << "/" << f2.getB();

	Fractie f3 = f1 + f2, f4 = f1 - f2, f5 = f1*f2, f6 = f1 / f2;

	cout << "\n\nFractia suma: " << f3.getA() << "/" << f3.getB();
	cout << "\nFractia diferenta: " << f4.getA() << "/" << f4.getB();
	cout << "\n\nFractia inmultire: " << f5.getA() << "/" << f5.getB();
	cout << "\nFractia impartire: " << f6.getA() << "/" << f6.getB() << endl << endl;

	cout << "\nFractii ext: \n";
	cout << "Fractia 1:\n  a=";
	cin >> nr;
	cout << "  b=";
	cin >> nm;
	Fractie_ext F1(nr, nm);

	cout << "\nFractia 2:\n  a=";
	cin >> nr;
	cout << "  b=";
	cin >> nm;
	Fractie_ext F2(nr, nm);

	return 0;
}

//header
//clasa Fractie
#pragma once

class Fractie
{
protected:
	int a, b;
```

```
public:
	Fractie()
	{
		a = 0;
		b = 1;
	}
	Fractie(int nr, int nm)
	{
		if (b != 0)
		{
			a = nr;
			b = nm;
		}
		else
			cout << "\nNumitorul trebuie sa fie diferit de zero!\n\n";
	}

	void setA(int val) { a = val; }
	void setB(int val) { b = val; }

	int getA() { return a; }
	int getB() { return b; }

	void simplificare(Fractie &f);

	Fractie operator+(Fractie f);
	Fractie operator-(Fractie f);
	Fractie operator*(Fractie f);
	Fractie operator/(Fractie f);
};

void Fractie::simplificare(Fractie &f)
{
	int nr = f.a, nm = f.b;

	while (nr != nm)
	{
		if (nr > nm)
			nr -= nm;
		else
			nm -= nr;
	}

	f.a /= nr;
	f.b /= nr;
}

Fractie Fractie::operator+(Fractie f)
{
	Fractie sum;

	sum.a = f.a*b + a * f.b;
	sum.b = f.b*b;

	simplificare(sum);

	return sum;
}
```

```
Fractie Fractie::operator-(Fractie f)
{
	Fractie dif;

	dif.a = a * f.b - f.a*b;
	dif.b = f.b*b;

	simplificare(dif);

	return dif;
}

Fractie Fractie::operator*(Fractie f)
{
	Fractie prod;

	prod.a = f.a*a;
	prod.b = f.b*b;

	simplificare(prod);

	return prod;
}

Fractie Fractie::operator/(Fractie f)
{
	Fractie div;

	div.a = f.b*a;
	div.b = f.a*b;

	simplificare(div);

	return div;
}

//clasa Fractie_ext
#pragma once

class Fractie_ext :public Fractie
{
public:
	Fractie_ext(int nr,int nm):Fractie()
	{
		setA(nr);
		setB(nm);
	}

	Fractie_ext& operator++()
	{
		this->setA(getA() + getB());
		this->setB(getB());

		simplificare(*this);

		return *this;
	}
```

```
        Fractie_ext& operator--()
        {
                this->setA(getA() - getB());
                this->setB(getB());

                simplificare(*this);

                return *this;
        }
};
```