Un al treilea exercitiu din acest laborator isi doreste sa reuneasca elementele din laboratoarele 1 , 2 si 3 cu o mostenire simpla:

Practic , materia pentru testul de laborator va fi:

- Creare clasa de baza avand:
  - Date membre private: **atribute**
  - Date membre publice
    - **Setter**
    - **Getter**
    - **Constructor implicit**
    - **Constructor cu parametri**
    - **Constructor de copiere**
    - **Destructor**
    - **Alte metode specifice clasei**
  - Date non-membre ale clasei
    - **Functii cu argument obiect**
    - **Functii cu argument referinta la obiect**
    - **Functii Friend**
- Creare clasa derivata avand:
  - Date membre private: **atribute**
  - Date membre publice
    - **Setter**
    - **Getter**
    - **Alte metode specifice clasei**
- **Creare functie main() in care:**
  - Cream obiecte ale clasei de baza care isi iau datele :
    - Din constructorul implicit
    - Din constructorul cu parametri
    - Prin copierea dintr-un obiect existent
  - Accesam functiile membre si non-membre pentru obiectele create
  - Cream obiecte de tipul clasei derivate si accesam metodele celor doua clase pentru aceste obiecte.

De mentionat ca intr-un subiect nu se vor gasi toate aceste elemente, ci doar o selectie din ele astfel incat sa se poata rezolva intr-un timp de 50 de minute.

Pentru a evidentia toate aceste elemente , construim clasa Patrat din care vom deriva clasa Piramida.

Explicatiile sunt notate sub forma de comentarii in codul sursa:

```
#include <iostream>
#include <math.h>

using namespace std;

class PATRAT
{//atributele vor fi private
	int latura;//sunt atribute private
	//pentru ca nu dorim sa fie modificate direct
	//private la atribute
	string culoare;
	public:
		//iar metodele vor fi publice
		void SetLatura(int dim);//setterul este in general void
		//pentru ca nu returneaza nimic
		void SetCuloare(string clr);
		int GetLatura();//getterul care returneaza o variabila
		//trebuie sa aiba ca tip , exact tipul variabilei returnate
		string GetCuloare();
		//urmeaza constructorii
		//constructorii nu au tip returnat!
		//constructorii au intotdeauna numele clasei !!!
		PATRAT();//declaratia (prototipul) constructorului implicit
		PATRAT(int dim, string clr);//declaratia (prototipul) constructorului cu
parametri
		//un constructor cu parametri nu ne nevoie sa aiba ca numar de parametri
		//exact numarul de atribute
		//unele atribute pot fi atribuite prin setter
		//pot folosica parametri aceleasi variabile ca la Setteri pentru ca
		//dim si clr au vizibilitate locala (nu sunt variable globale)
		PATRAT(const PATRAT &p);//declaratia constructorului de copiere
		//primeste ca parametru o referinta constanta de tipul PATRAT
		~PATRAT();//declaratia destructorului


		friend int Perimetru(PATRAT *P);
		//functia friend pentru clasa PATRAT
			//functiile friend au acces la toate datele membre ale clasei
			//inclusiv cele PROTECTED si PRIVATE
			//daca stergem cuvantul cheie rezervat friend de la functia
Perimetru
			//aceasta nu mai are acces direct prin sageata -> la atributul
privat latura

		int Arie();

		//functiile Generic1 si Generic2 nu se declara in clasa
		//pentru ca nu sunt functii membre ale clasei

};//clasa se inchide mereu cu punct si virgula

//inafara clasei definesc metodele clasei PATRAT accesandule cu operatorul de
rezolutie
```

```
void PATRAT::SetLatura(int dim)//definitie (implementare) setter pentru latura
{
	latura=dim;
}

void PATRAT::SetCuloare(string clr)//definitie (implementare) setter pentru culoare
{
	culoare=clr;
}

int PATRAT::GetLatura()//definitie (implementare) getter pentru latura
{//returneaza intotdeauna atribute , nu variabile locale (dim)
	return latura;
}

string PATRAT::GetCuloare()//definitie (implementare) getter pentru culoare
{//returneaza intotdeauna atribute , nu variabile locale (clr)
	return culoare;
}

PATRAT::PATRAT()//definitie (implementare) constructor implicit
{//constructorul implicit atribuie valori default obiectelor
//care sunt create fara parametri
	latura=10;
	culoare="verde";
	cout<<"S-a creat un obiect implicit"<<endl;
}

PATRAT::PATRAT(int dim, string clr)//definitie (implementare) constructor cu
parametri
{//constructorul cu parametri atribuie valorile variabilelor date ca parametri
//este folosit de obiectele create cu valori efective
	latura=dim;
	culoare=clr;
	cout<<"S-a creat un obiect cu parametri"<<endl;
}

PATRAT::PATRAT(const PATRAT &p)
{//fiecare atribut al obiectului nou preia (latura)
//valoarea corespunzatoare a aceluiasi atribut din obiectul vechi(p.latura)
	latura=p.latura;
	culoare=p.culoare;
	cout<<"S-a copiat un obiect"<<endl;

}

PATRAT::~PATRAT()
{//destructor care sterge obiectele din memorie cand
//nu mai sunt folosite de metodele clasei
//si afiseaza un mesaj
	cout<<"S-a sters un obiect"<<endl;
}
```

```
//functie care NU este membra a clasei deci nu se acceseaza cu operatorul de
rezolutie
//nu se mai foloseste cuvantul cheie rezervat friend inafara clasei
int Perimetru(PATRAT *P)//definitia functiei friend
{//care calculeaza perimetrul
	int perimetru;
	perimetru=4*P->latura;//P fiind pointer se acceseaza cu operatorul sageata ->
	//latura este private, functia friend are acces la datele PROTECTED si PRIVATE
	return perimetru;
}

//este functie membra a clasei deci o accesam cu operatorul de rezolutie
int PATRAT::Arie()
{//care calculeaza aria
	int arie;
	arie=pow(latura,2);
	return arie;
}

//Generic1 nu se acceseaza cu operatorul de rezolutie :: pentru ca
//pentru ca nu este functie membra a clasei
void Generic1(PATRAT P)//primeste ca parametru un obiect de tipul PATRAT
{//copie pe stiva intr-o alta locatie de memorie datele obiectului primit ca
parametru
//deci apeleaza constructorul de copiere adica
//se afiseaza INTAI mesajul din contructorul de copiere
	cout<<"S-a folosit functia Generic1"<<endl;
	//APOI se afiseaza mesajul din Generic1
}

//Generic2 nu se acceseaza cu operatorul de rezolutie :: pentru ca
//pentru ca nu este functie membra a clasei
PATRAT Generic2(PATRAT &P)//primeste ca parametru o referinta la un obiect de tipul
PATRAT
{//nu mai copie pe stiva, ci doar pastreaza adresa data prin referinta
	cout<<"S-a folosit functia Generic2"<<endl;
	//se afiseaza INTAI mesajul din Generic2
	return P;
	//dupa care se copie pe stiva intr-o alta locatie de memorie datele obiectului
primit ca referinta
	//deci apeleaza constructorul de copiere adica
	//se afiseaza APOI mesajul din contructorul de copiere
}
```

```
//creez clasa derivata PIRAMIDA
//PIRAMIDA mosteneste datele membre din PATRAT
class PIRAMIDA : public PATRAT// se foloseste operatorul : , nu operatorul ::
{
	int inaltime;
	public://declar si definesc in interiorul clasei Setterul, Getterul si functia proprie Volum
		void SetInaltime(int dim)
		{
			inaltime=dim;//pot folosi din nou dim pentru ca e variabila locala
		}
		int GetInaltime()
		{
			return inaltime;
		}
		int Volum()
		{
			int volum;
			volum=Arie()*inaltime/3;
			//metoda Volum are acces la metoda Arie pentru ca mostenirea
			//public-public permite acest lucru
			return volum;
		}

};//si clasa derivata trebuie inchisa tot cu punct si virgula ;



int main()
{	//creez intai 4 obiecte de tip PATRAT (clasa de baza)
	PATRAT P1;//creez un obiect P1 care ia datele din constructorul implicit
	cout<<"Patratul P1 este de culoare "<<P1.GetCuloare()<<", are latura de "<<P1.GetLatura()<<" metri, perimetrul de "<<Perimetru(&P1)<<" metri si aria de "<<P1.Arie()<<" metri patrati"<<endl ;
	//la functia friend Perimetru , argumentul il oferim ca adresa (referinta) pentru ca la declaratie aceasta functie primea acest argument ca pointer

	PATRAT P2(15, "albastru");//creez un obiect P2 care ia datele din constructorul cu parametri
	cout<<"Patratul P2 este de culoare "<<P2.GetCuloare()<<", are latura de "<<P2.GetLatura()<<" metri, perimetrul de "<<Perimetru(&P2)<<" metri si aria de "<<P2.Arie()<<" metri patrati"<<endl ;

	PATRAT P3=P1;//creez un obiect P3 care copie datele obiectului P1; sintaxa standard
	cout<<"Patratul P3 este de culoare "<<P3.GetCuloare()<<", are latura de "<<P3.GetLatura()<<" metri, perimetrul de "<<Perimetru(&P3)<<" metri si aria de "<<P3.Arie()<<" metri patrati"<<endl ;

	PATRAT P4(P2);//creez un obiect P4 care copie datele obiectului P2; sintaxa alternativa;
	cout<<"Patratul P4 este de culoare "<<P4.GetCuloare()<<", are latura de "<<P4.GetLatura()<<" metri, perimetrul de "<<Perimetru(&P4)<<" metri si aria de "<<P4.Arie()<<" metri patrati"<<endl ;
```

```
	Generic1(P3);//intai apare mesajul din constructorul de copiere apoi mesajul
din Generic1
	Generic2(P4);//intai apare mesajul din Generic2 apoi mesajul din constructorul
de copiere(invers ca la Generic1)
	//functiile Generic1 si Generic2 reprezinta o modalitate de a folosi
constructorul de copiere, nu doar la crearea altor obiecte
	//ci si la copierea datelor din obiectele existente in alte locatii de memorie
decat cele initiale

	//creez acum un obiect de tipul clasei derivate PIRAMIDA
	PIRAMIDA PD1;//desi nu am constructori in clasa derivata, compilatorul imi va
oferi unul implicit
	//pentru ca nu am alti constructori declarati explicit in clasa PIRAMIDA
	PD1.SetLatura(25);//obiectul de tip piramida are acces la Setterii din clasa
PATRAT
	PD1.SetCuloare("galben");
	PD1.SetInaltime(5);
	cout<<"Piramida PD1 are baza de culoare "<<PD1.GetCuloare()<<", latura bazei
de "<<PD1.GetLatura()<<" metri, inaltimea de "<<PD1.GetInaltime()<<" metri, "<<endl;
	cout<<" perimetrul de "<<Perimetru(&PD1)<<" metri, aria bazei de
"<<PD1.Arie()<<" metri patrati si volumul "<<PD1.Volum()<<" metri cubi ."<<endl ;
	//Perimetru(&PD1) nu da eroare deoarece la functia friend au au acces atat
obiectele clasei de baza cat si obiectele clasei derivate


}
```