PROGRAMARE OBIECT-ORIENTATA

LABORATOR 2 – CLASE SI OBIECTE (CONTINUARE) , CONSTRUCTOR CU PARAMETRI, CONSTRUCTOR DE COPIERE. FUNCTII FRIEND.

In laboratorul anterior am vazut ca pentru a initializa cu valori atributele unei clase putem folosi pe de o parte constructorul implicit iar pe de alta parte metodele speciale de tip “Setter”. In aceasta lucrare vom explica un alt tip de constructor prin care putem aloca valori atributelor si anume “constructorul cu parametri”.

Intr-o alta ordinde de idei , indiferent prin ce metoda alegem sa alocam valori atributelor, modalitatea prin care returnam / ne folosim de valorile atributelor este prin “Getter”.

Exemplul 1: Modelam un nou tip de date denumit “Pistol”:

Acest nou tip de date poate avea urmatoarele atribute:

- ✓ Numarul maxim de gloante care va fi o variabila de tip int
- ✓ Numarul curent de gloante, tot variabila de tip int, dar care poate fi intre zero si numarul maxim.
- ✓ O piedica cu care putem bloca pistolul, variabila de tip Boolean intrucat nu poate avea decat 2 stari (piedica pe ON sau OFF)

Din punct de vedere al metodelor vom crea :

- ✓ Cate un setter si un getter pentru numar maxim de gloante si piedica
- ✓ Getter pentru numarul curent de gloante
- ✓ Un constructor implicit in care intializam numarul maxim de gloante si starea piedicii.
- ✓ Un constructor cu parametri
    - ➢ La crearea obiectelor , compilatorul analizeaza argumentele; daca obiectul se creaza fara argumente atunci datele sunt luate din constructorul implicit. Orice obiect creat fara argumente , va lua datele din constructorul implicit
    - ➢ Daca obiectul are parametrii precizati atunci acestia trebuie sa corespunda ca numar cu cei din constructorul cu argumente deja creat; altfel vom primi eroare
    - ➢ Conceptul pe care se bazeaza compilatorul in alegerea tipului de constructor in functie de existenta si numarul de parametri se numeste POLIMORFISM sau SUPRAINCARCAREA FUNCTIILOR (exista in programare si SUPRAINCARCAREA operatorilor)

- ✓ Metoda Trage() in care verificam intai starea piedicii ; doar daca piedica este OFF verificam daca mai avem gloante si apoi putem trage cu pistolul care echivaleaza cu o decrementare a numarului curent de gloante. Vom da si un mesaj “BOOM !” in momentul cand facem actiunea propriu-zisa de a trage cu pistolul.

- ✓ Metoda Incarca() o apelam cand nu mai avem gloante. La aceasta verificam intai daca nu cumva avem deja pistolul incarcat. Daca avem mai putin decat numarul maxim e gloante atunci putem incarca.
- ✓ Metoda Pune piedica, aceasta verifica intai daca nu numva este deja pusa piedica si apoi seteaza piedica pe ON
- ✓ Metoda Scoate Piedica , in mod similar verificam daca piedica este deja scoasa si abia apoi scoatem piedica.

Sa analizam implementarea exemplului 1:

```
#include <iostream>
using namespace std;

class Pistol //definesc un nou tip de date
{
	int NrMaxGloante;
	int NrCrtGloante;
	bool Piedica;

	public:
		void SetNrMaxGloante(int Max); //setter pentru NrMaxGloante
		void SetPiedica(bool p); //setter pentru Piedica
				//NRCrtGloante nu are nevoie neaparat de setter pentru ca
				//va fi incrementat sau decrementat in functie de cate ori
				//incarcam pistolul sau tragem cu pistolul
		int GetNrMaxGloante(); //getter pentru NrMaxGloante
		int GetNRCrtGloante(); //getter pentru NRCrtGloante
		bool GetPiedica(); //getter pentru Piedica

		void Trage();
		void Incarca();
		void PunePiedica();
		void ScoatePiedica();

		Pistol();//constructor implicit
		Pistol(int NrMAX, bool pd);//constructor cu argumente

};

void Pistol::SetNrMaxGloante(int Max) //setter pentru NrMaxGloante
{
	NrMaxGloante=Max;
}

void Pistol::SetPiedica(bool p) //setter pentru Piedica
{
	Piedica=p;
}

int Pistol::GetNrMaxGloante() //getter pentru NrMaxGloante
{
	return NrMaxGloante;
```

```
}

int Pistol::GetNRCrtGloante() //getter pentru NRCrtGloante
{
	return NrCrtGloante;
}

bool Pistol::GetPiedica() //getter pentru Piedica
{
	return Piedica;
}

void Pistol::Trage()//implementare metoda trage
{
	if(Piedica==true)//verific daca piedica este pusa
	{
		cout<<"Piedica este pusa, nu se poate trage";
	}
	else//daca piedica nu e pusa verific nr de gloante
		{
		if(NrCrtGloante>0)//verific daca am gloante
			{
				//trag un glont adica decrementez nr de gloante existent
	NrCrtGloante--;
				cout<<"BOOM !!!";//trag cu pistolul
			}
			else
				{	//daca nu mai am gloante afisez doar un mesaj
					cout<<"Nu mai sunt gloante !!!";
				}
		}
}

void Pistol::Incarca()
{//verific intai daca e deja incarcat
	if (NrCrtGloante==NrMaxGloante)
	{
		cout<<"Pistolul este deja incarcat !!!";
	}
	else//daca are mai putine gloante decat maxim
		{
			NrCrtGloante=NrMaxGloante;//incarc pistolul
		}
}









void Pistol::PunePiedica()
{//verific daca piedica e deja pusa
```

```
	if(Piedica==true)
	{
		cout<<"Piedica este deja pusa";
	}
	else//daca nu e pusa atunci setez pe true
		{
			Piedica=true;
		}
}

void Pistol::ScoatePiedica()
{//verific daca piedica e deja scoasa
	if(Piedica==false)
	{
		cout<<"Piedica este deja scoasa";
	}
	else//daca nu se scoasa setez pe false valoarea piedicii
		{
			Piedica=false;
		}
}


Pistol::Pistol()
{//initializez valorile atributelor in constructor implicit
	cout<<"Am preluat datele din constructorul implicit\n";
	NrMaxGloante=5;
	Piedica=false;
}

Pistol::Pistol(int NrMAX, bool pd)
{//initializez valorile atributelor in constructor cu 2 argumente
	cout<<"Am preluat datele din constructorul cu parametri\n";
	NrMaxGloante=NrMAX;
	Piedica=pd;
}

int main()
{
	Pistol p0;//creez un obiect de tip pistol(fara argumente) deci ia datele din
constructorul implicit)
	if(p0.GetPiedica()==true)//verific daca pistolul creat are piedica pusa
	{//afisez mesaje diferite pe ecran in functie de piedica
		cout<<"Postolul implicit p0 are maxim "<<p0.GetNrMaxGloante()<<" gloante si
piedica pusa\n\n";
	}
	else{
			cout<<"Postolul implicit p0 are maxim "<<p0.GetNrMaxGloante()<<" gloante
si piedica deblocata\n\n";
		}


	Pistol p1(10,true);//creez un alt obiect de tip pistol(cu argumente) deci ia datele
din constructorul implicit)
	if(p1.GetPiedica()==true)
```

```
	{//afisez mesaje diferite in functi de starea piedicii
		cout<<"Postolul p1 are maxim "<<p1.GetNrMaxGloante()<<" gloante si piedica
pusa\n\n";
	}
	else{
			cout<<"Postolul p1 are maxim "<<p1.GetNrMaxGloante()<<" gloante si
piedica deblocata\n\n";
		}
}
```

Vom analiza in alt exemplu de cod in care modelam clasa **Masina**.

Aceasta are atributele **private** culoare si capacitatea rezervorului. Ca metode special sunt construiti setteri si getteri pentru fiecare din aceste doua atribute. Metodele sunt declarate in clasa ca si prototip dar sunt definite inafara clasei avand in fata numele clasei din care apartin folosind operatorul **::** de rezolutie pentru a descrie apartenenta la o anume clasa.

In functia main cream mai multe obiecte de tip masina. Unele din acestea preiau valorile din constructorul implicit, altele preiau valorile din constructorul cu parametri. Decizia se ia prin analiza din partea compilatorului a numarului dar si a tipului de parametric(argumente). Decizia se poate lua datorita prezentei conceputului de Polimorfism de functii din C++.

Codul clasei masina este urmatorul:

```
#include <iostream>
using namespace std;

class Masina
{//atributele private ale clasei Masina
	string culoare;
	int rezervor;

	public:
		void SetCuloare(string c);//setter pentru culoare
		void SetRezervor(int r);//Setter pentru capacitatea rezervorului

		string GetCuloare();//getter pentru culoare
		int GetRezervor();//getter pentru capacitatea rezervorului

		Masina();//constructor implicit
		Masina(string cul, int cap);//constructor cu parametri

};

void Masina::SetCuloare(string c)
{
	culoare=c;
}
```

```
void Masina::SetRezervor(int r)
{
	rezervor=r;
}

string Masina::GetCuloare()
{
	return culoare;
}

int Masina::GetRezervor()
{
	return rezervor;
}

Masina::Masina()//initializarea constructorului implicit
{//folosesc operatorul de rezolutie pentru a preciza apartenenta
//constructorului la o clasa anume
	culoare="alb";
	rezervor=20;
}

Masina::Masina(string cul, int cap)//initializarea constructorului cu parametri
{//constructorii nu au tip returnat , nici macar void !!!
	culoare=cul;
	rezervor=cap;
	cout<<"s-a apelat constructorul cu parametri"<<endl;
}

int main()
{
	Masina m0;//creez un obiect de tip masina care isi ia datele din constructorul
implicit
	//ii afisez caracteristicile
	cout<<"Masina implicita m0 are culoarea "<<m0.GetCuloare()<<" si capacitatea
rezervorului de "<<m0.GetRezervor()<<" litri "<<endl;

	Masina m1;//creez un alt obiect care isi va lua datele din setteri
	m1.SetCuloare("galben");
	m1.SetRezervor(40);
	cout<<"Masina m1 are culoarea "<<m1.GetCuloare()<<" si capacitatea rezervorului de
"<<m1.GetRezervor()<<" litri "<<endl;


	Masina m2("rosu",35);//datele sunt luate din constructorul cu 2 argumente
	cout<<"Masina m2 are culoarea "<<m2.GetCuloare()<<" si capacitatea rezervorului de
"<<m2.GetRezervor()<<" litri "<<endl;
	//prin orice metoda as seta valorile unui obiect , getterii sunt cei cu care pot afisa
acele valori.
	return 0;

}
```

In continuare dorim sa extindem clasa Masina si sa ii adaugam un constructor de copiere.
Acesta are rolul de a prelua valorile atributelor pentru un obiect existent si de a le copia ca valori ale unui atribut nou. Obiectul sursa si obiectul destinatie trebuie sa fie de acelasi tip(din aceiasi clasa)
Vom face apoi o functie friend in care comparam datele din cele doua obiecte (cel sursa si cel destinatie)

```
#include <iostream>
using namespace std;

class Masina
{
	string culoare;
	int rezervor;

	public:
		void SetCuloare(string c);
		void SetRezervor(int r);

		string GetCuloare();
		int GetRezervor();

		Masina();	//constructor implicit
		Masina(string cul, int cap);	//constructor cu parametri

		friend bool ComparaObiect(const void*, const void *);// functia prietena,
construita pentru a putea fi preluata ca parametru

		Masina(const Masina&);//constructor de copiere

};

void MesajDecizie(unsigned int*, Masina*, Masina*, bool (*pFctCompare)(const void *a, const
void *b));	// functia de decizie: prototip

void Masina::SetCuloare(string c)
{
	culoare=c;
}

void Masina::SetRezervor(int r)
{
	rezervor=r;
}

string Masina::GetCuloare()
{
	return culoare;
}

int Masina::GetRezervor()
```

```
{
	return rezervor;
}

bool ComparaObiect(const void *pM1, const void *pM2)
// definitia functiei friend, fara scope resolution (interzis)
{
	return ((*(Masina*)pM1).GetCuloare() == (*(Masina*)pM2).GetCuloare()) ? true : false;
	// aici am nevoie de getteri. In plus am conversia explicita a pointerului 'void*' la
(Masina*)
}

Masina::Masina()
// definitia constructorului implicit
{
	culoare="alb";
	rezervor=20;
}

Masina::Masina(string cul, int cap)
// definitia constructorului cu parametri
{
	culoare=cul;
	rezervor=cap;
	cout<<"s-a apelat constructorul cu parametri"<<endl;
}

Masina::Masina(const Masina& m1)  // definitia constructorului de copiere
{
	cout<<"constructor de copiere" << endl;
	culoare	= m1.culoare;
	rezervor = m1.rezervor;
}


int main()
{
	Masina m0;
	cout<<"Masina implicita m0 are culoarea "<<m0.GetCuloare()<<" si capacitatea
rezervorului de "<<m0.GetRezervor()<<" litri "<<endl;

	Masina m1;
	m1.SetCuloare("galben");
	m1.SetRezervor(40);
	cout<<"Masina m1 are culoarea "<<m1.GetCuloare()<<" si capacitatea rezervorului de
"<<m1.GetRezervor()<<" litri "<<endl;

	Masina m2("rosu",35);
	cout<<"Masina m2 are culoarea "<<m2.GetCuloare()<<" si capacitatea rezervorului de
"<<m2.GetRezervor()<<" litri "<<endl;

	Masina m3 = m2;
	cout<<"Masina m3 are culoarea "<<m3.GetCuloare()<<" si capacitatea rezervorului de
"<<m3.GetRezervor()<<" litri "<<endl;
```

```
	// utilizare functie friend
	unsigned int decizie = 2;  // am ales special o valoare diferita de True False pe care
le-am dorit de la inceput
	cout << endl;

	cout << "Decizia inainte de a face comparatia obiectelor este: " << decizie << endl;
	Masina *pMas0 = &m0, *pMas2 = &m2;


	MesajDecizie(&decizie, pMas0, pMas2, ComparaObiect);
	cout << "Decizia ulterior compararii este " <<  decizie << ", adica: " << ((decizie) ?
"Aceeasi culoare" : "Culori diferite") << endl;

	return 0;
}

void MesajDecizie(  unsigned int *pDecizie,
                                Masina *pM1,
                                Masina *pM2,
                                bool (*pFctCompare)(const void *a, const void *b)
                            )
{
	if( 1 == pFctCompare((Masina*)pM1, (Masina*)pM2) )
		*pDecizie = true;
	else
		*pDecizie = false;
}
```