**Exercitii Tutoriat 3**

→ friend, const, static, redefinirea operatorilor

**Cerinta: Pentru fiecare dintre programele de mai jos, spuneți dacă sunt corecte. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți ce nu este corect , de ce și ce ați corecta.**

**Exercitiul 1**

**#include <iostream>**

**using namespace std;**

**class Test**

**{**

**static int x;**

**public:**

**Test (int i)**

**{**

**x+=i;**

**cout << x <<' ';**

**}**

**};**

**static int x = 0;**

**int main()**

**{**

**Test t [] ={Test(5), Test(24), Test(6)};**

**return 0;**

**}**

**Nu compileaza: “ undefined reference to `Test::x’ “**

**Variabila x din clasa Test este statica, deci trebuie initializata in afara clasei, in mod corespunzator. Acel static int x=0; (linia 14) se refera pur si simplu la o variabila globala statica, nu la x din clasa.**

**Modificare : linia 14: int Test:: x;**

**Exercitiul 2**

**In plus, ce valoare va avea x la finalul executiei programului?**

**#include <iostream>**

**using namespace std;**

**int static x;**

**class Test1**

**{**

**public:**

**Test1 () {cout<<"Test1"<<endl; x++; }**

**~Test1() { cout<<"Destructor Test1"<<endl; x--;}**

**};**

**class Test2**

**{**

**Test1 test;**

**public:**

**Test2(){cout<<"Test2"<<endl; x++; }**

**static void afisare() {cout<<"Static Test2"<<endl; x\*=2;}**

**};**

**int main()**

**{**

**Test2 test;**

**test.afisare();**

**cout<<x<<endl;**

**return 0;**

**}**

**Compileaza:**

**Test1**

**Test2**

**Static Test2**

**4**

**Destructor Test1**

**Explicatie:**

**Test1 → obiectul din clasa Test2 (compunere); se apleaza constructorul pentru test si x=1**

**Test2 → constructorul pentru obiectul test din main si x=2**

**Static Test2 → functia statica din clasa Test2 si x=4**

**Se afiseaza x=4**

**Destructor Test1 → la finalul programului se distrug obiectele ( deci apelam destructorul pentru Test1 test al clasei Test2)**

**Obs: in final x va avea valoarea 3 → din apelul destructorului**

**Exercitiul 3**

**#include <iostream>**

**using namespace std;**

**class Test**

**{**

**Test(){cout<<"Test ";}**

**~Test(){cout<<"~Test ";}**

**friend Test functie();**

**};**

**Test functie() { return Test(); }**

**int main()**

**{**

**Test test = functie();**

**return 0;**

**}**

**Nu compileaza: Destructorul clasei Test este privat**

**“Test::~Test() noexcept" (declared at line 9) is inaccessible”**

**Explicatie:**

**-Se apeleaza metoda functie a clasei Test → este ok, se poate apela**

**-In interiorul ei se apeleaza constructorul Test() → este ok : functie e friend cu Test, deci -are acces la membri sai privati**

**-Eroarea apare cand vrem sa distrugem obiectul : destructorul este privat, deci nu poate fi accesat in main**

**OBS: daca nu am fi redefinit destructorul, nu am fi avut eroare la compilare…**

**Modificare: Schimbam modificatorul de acces din private (default) in public**

**In acest caz, ce se afiseaza la stdout (adica in consola)?**

**Exercitiul 4**

**#include <iostream>**

**using namespace std;**

**class Test**

**{**

**int varsta;**

**public:**

**Test(): varsta(1){}**

**void sum(Test& s) const { cout << varsta + s.varsta; }**

**};**

**int main()**

**{**

**Test s1;**

**const Test s2;**

**s1.sum(s2);**

**return 0;**

**}**

**Nu compileaza:**

**Obiectul s2 este constant. Apare eroare in momentul in care incearca sa apeleze metoda sum, deoarece l-am transmis prin referinta, deci poate fi modificat (ceea ce incalca principiile const-ului).**

**Compilatorul nu verifica în interiorul metodei / functiei daca obiectul chiar este modificat sau nu. El cauta cuvantul cheie care să garanteze că va ramane constant si daca nu ill gaseste intoarce o eroare.**

**Modificare:**

**Varianta 1: modificam antetul functiei, declarand obiectul const ⇒ nu va putea fi modificat in functie**

**void sum(const Test& s) const { cout << varsta + s.varsta; }**

**Varianta 2: nu mai transmitem obiectul prin referinta ⇒ originalul nu va putea fi modificat pentru ca transmitem doar o copie a sa.**

**void sum(Test s) const { cout << varsta + s.varsta; }**

**OBS: daca nu am fi redefinit constructorul default am fi avut eroare : const variable "s2" requires an initializer -- class "Test" has no user-provided default constructor.**

**Exercitiul 5**

**#include <iostream>**

**using namespace std;**

**class Test**

**{**

**const int x = 10;**

**public:**

**Test() : x(4){}**

**const int getX() const {return x;}**

**};**

**int main()**

**{**

**Test \*obj = new Test();**

**cout << obj->getX();**

**return 0;**

**}**

**Compileaza: 4**

**Cu toate ca am dat o valoare constantei x in definitia clasei, aceasta poate fi modificata prin intermediul listei de initializare a constructorului.**

**Pointeul \*obj de tip Test poate apela fara probleme metoda getX care e constanta**

**Exercitiul 6**

**#include <iostream>**

**using namespace std;**

**class Cls**

**{**

**int x;**

**public:**

**Cls(int i) : x(i) {}**

**const int& f() {return x;}**

**};**

**int main()**

**{**

**Cls a(14);**

**int b = a.f()++;**

**cout << b << '\n';**

**return 0;**

**}**

**Nu compileaza:**

**expression must be a modifiable lvalue**

**increment of read-only location 'a.Cls::f()'**

**Explicatie:**

**a.f()++ → se apeleaza functia f() a clasei Cls care returneaza o referinta constanta de tip int ⇒ nu putem sa modificam rezultatul ( adica sa efectuam operatii direct asupra lui a.f())**

**Modificare:**

**Varianta 1: nu modificam rezultatul . adica nu incrementam rezultatul primit in urma apelului a.f()**

**int b = a.f();**

**Varianta 2: returnam o referinta neconstanta, deci putem modifica rezultatul (adica sa facem a.f() ++ );**

**int&f() {return x;}**

***Exercitiul 7***

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**int x;**

**static int y;**

**public:**

**A(int i,int j):x(i),y(j){}**

**int f() const { return y;}**

**};**

**int A::y;**

**int main()**

**{**

**const A a(21,2);**

**cout<<a.f();**

**return 0;**

**}**

**Nu compileaza:**

**"y" is not a nonstatic data member or base class of class "A"**

**'int A::y' is a static data member; it can only be initialized at its definition**

**În acest cod, încercăm să inițializăm y în lista de inițializare a constructorului (y(j)). Acest lucru nu este permis, deoarece y este un membru static, iar membrii statici nu pot fi inițializați în constructorii de instanță.**

**Deci, daca vrem sa facem o initializare:**

**Constantele → lista de initializare a constructorului**

**Static → in corpul constructorului**

**Modificare:**

**A(int i,int j):x(i){ y=j;} // se afiseaza apoi 2**

**Exercitiul 8**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**const int x;**

**static int y;**

**public:**

**A(int i,int j):x(++y){}**

**int f() const { return y;}**

**};**

**int A::y;**

**int main()**

**{**

**const A a(1,2);**

**cout<<a.f();**

**return 0;**

**}**

**Compileaza: 1**

**se incrementează y (++y), iar rezultatul este folosit pentru inițializarea lui x. Acest lucru este permis deoarece y este o variabilă statică, partajată între toate instanțele clasei, iar operațiile asupra acesteia nu depind de o anumită instanță a clasei.**

**Exercitiul 9**

**#include <iostream>**

**class cls**

**{**

**int x;**

**public:**

**cls(int y) {x=y; }**

**friend int operator\*(cls a,cls b){return (a.x\*b.x); }**

**};**

**int main()**

**{**

**cls m(5),n(24);**

**std::cout << m\*n;**

**return 0;**

**}**

**Compileaza: 120 ( adica 5\*24 )**

**Exercitiul 10**

**#include <iostream>**

**using namespace std;**

**class Test**

**{**

**int vi;**

**public:**

**Test(int v = 28) { vi = v; }**

**operator int() { return vi; }**

**Test operator++()**

**{ cout<<"operator++() "; vi++; return \*this; }**

**Test operator++(int);**

**};**

**Test Test::operator++(int)**

**{**

**cout<<"operator++(int) ";**

**Test aux = \*this;**

**vi++;**

**return aux;**

**}**

**int main()**

**{**

**Test p(2);**

**int x = p++, y = ++p;**

**cout<<endl;**

**cout << x <<' ' <<y << endl;**

**return 0;**

**}**

**Compileaza:**

**operator++(int) operator++()**

**2 4**

**Explicatie:**

**-se apeleaza constructorul pentru obiectul p ⇒ p.vi=2**

**-pentru ca in clasa Test avem o singura data membra de tip int, compilatorul va face o conversie automata de la Test la int**

**-x = p++ ⇒ x = 2 apoi p= 3 → apelam operator++(int)**

**-y = ++p ⇒ p = 4 apoi y = 4 → apelam operator++()**

**i++ vs ++i**

**i++ → se creaza o variabila auxiliara in memorie→ intai il folosim pe i, apoi il incrementam**

**++i → nu se creaza o variabila auxiliara → intai il incrementam pe i, apoi il foloism**

**De accea pentru opearorul++ cu antetul operator++(int) este nevoie de acea variabila auxiliara**

**Exercitiul 11**

**#include <iostream>**

**class Test**

**{**

**int \*v, nr;**

**public:**

**Test(int i)**

**{**

**nr = i;**

**v = new int[i];**

**for (int j = 1; j < nr; j++)**

**v[j] = 0;**

**}**

**int size() { return nr; }**

**int& operator[](int i) { return \*(v + i); }**

**};**

**int main()**

**{**

**Test x(10);**

**x[4] = -15;**

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

**std::cout << x[i]<<' ';**

**return 0;**

**}**

**Compileaza: -1163005939 0 0 0 -15 0 0 0 0 0 (primul numar e o valoare random din memorie)**

**Explicatie:**

**-Se apeleaza constructul pentru obiectul x ⇒ nr = 10, dupa care declaram un array dinamic de 10 elemente.**

**-De la 1 pana la 9 in array-ul v , se pune pe pozitia corespunzatoare valoarea 0 ( pe pozitia 0 nu am pus nimic, deci ramane neinitializat → va avea o valoarea random din memorie)**

**In acelasi context, se apeleaza pentru fiecare element v[j] operatorul [ ] redefinit, iar in interiorul acestui operator , operatorul ( ).**

**Schema de apel pentru fiecare element : v[j] → operator [ ] → operator ( )**

**Exercitiul 12**

**#include <iostream>**

**#include <cstring>**

**using namespace std;**

**class A**

**{**

**private:**

**int x;**

**char \*s;**

**public:**

**A(int x, char \*s)**

**{**

**this->x = x;**

**this->s = new char[strlen(s) + 1];**

**strcpy(this->s, s);**

**}**

**A(const A &a)**

**{**

**x = a.x;**

**s = new char[strlen(a.s) + 1];**

**strcpy(s, a.s);**

**}**

**~A()**

**{ delete[] s; }**

**friend istream &operator>>(istream &out, A &a);**

**};**

**istream &operator>>(istream &out, A &a)**

**{**

**out << a.x;**

**out << a.s;**

**return out;**

**}**

**int main()**

**{**

**A a(10, "abc");**

**A b=a;**

**cout << b << " " << a;**

**Nu compileaza. Am realizat supraincarcarea operatorului de citire, nu a celui de afisare. Trebuie modificat operatorul declarat friend astfel:  
 ostream &operator<<(ostream &out, A &a)**

**Exercitiul 13**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**class A**

**{**

**private:**

**string str;**

**public:**

**A(string s)**

**{**

**str = s;**

**}**

**friend class B;**

**};**

**class B**

**{**

**public:**

**void print(A a)**

**{**

**cout << a.str;**

**}**

**};**

**int main()**

**{**

**A a("Hello");**

**B b;**

**a.print(b);**

**return 0;**

**}**

**Nu compileaza. Explicatie: metoda print este membra a clasei B, si are un parametru de tip A. Astfel, apelul ar trebui sa fie b.print(a)**

**Exercitiul 14**

**#include <iostream>**

**using namespace std;**

**class Test**

**{**

**int x;**

**public:**

**Test(int x = 0) : x(x) {}**

**};**

**class Test2**

**{**

**int x;**

**public:**

**Test2(int x = 0) : x(x) {}**

**friend void print(Test2& t);**

**};**

**void print(Test& t)**

**{ cout << t.x << endl; }**

**void print(Test2& t)**

**{ cout << t.x << endl; }**

**int main()**

**{**

**Test t(5);**

**Test2 t2(5);**

**print(t);**

**print(t2);**

**return 0;**

**}**

**Nu compileaza.**

**error: 'int Test::x' is private within this context**

**20 | { cout << t.x << endl; }**

**note: declared private here**

**6 | int x;**

**Rezolvare: declaram functia print friend in clasa Test**

**Exercitiul 15**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**public:**

**void print()**

**{**

**cout << "A ";**

**}**

**~A()**

**{ cout << "DA "; }**

**friend void print(A a);**

**};**

**class B**

**{**

**public:**

**void print(A a)**

**{**

**cout << "B ";**

**}**

**friend void print(B b);**

**};**

**void print(A a)**

**{**

**cout << "AA ";**

**}**

**void print(B b)**

**{**

**cout << "BB ";**

**}**

**int main()**

**{**

**A a;**

**B b;**

**int x = 5;**

**while (x<6)**

**{**

**A c;**

**c.print();**

**x++;**

**}**

**a.print();;**

**print(b);**

**return 0;**

**}**

**Compileaza. Se afiseaza A DA A BB DA.**

**Explicatie:**

**c.print() afiseaza A**

**la iesirea din while obiectul c se distruge => destructorul afiseaza DA**

**a.print() afiseaza A**

**print(B) afiseaza B**

**La finalul programului se distruge si obiectul a => se afiseaza DA**

**Exercitiul 16**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**public:**

**void afisare()**

**{ cout << "B"; }**

**};**

**class MyClass**

**{**

**int x;**

**public:**

**MyClass(int val) : x(val)**

**{ afisare(); }**

**MyClass()**

**{ x = 0;**

**afisare(1); }**

**void afisare()**

**{ cout << x; }**

**};**

**void afisare()**

**{**

**cout << "Hello ";**

**}**

**void afisare(int x)**

**{**

**B b;**

**b.afisare();**

**}**

**int main()**

**{**

**MyClass obj(10);**

**afisare();**

**return 0;**

**}**

**Nu compileaza. error: no matching function for call to 'MyClass::afisare(int)'|**

**note: candidate: 'void MyClass::afisare()'**

**note: candidate expects 0 arguments, 1 provided**

**Modificare: scoatem argumentul de la apelul afisare(1)**