**Exercitii Tutoriat 4**

→ Mostenire

**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.**



**Partea 1 - Exercitii usoare**

**Exercitiul 1**

#include<iostream>

using namespace std;

class A

{

public:

A(){cout<<"A ";}

A(int x) {cout<<"A("<<x<<") ";}

~A(){cout<<"~A ";}

};

class B: public A

{

public:

B(){cout<<"B ";}

B(int x) {cout<<"B("<<x<<") ";}

~B(){cout<<"~B ";}

};

class C :public B

{

public:

C(){cout<<"C ";}

C(int x) {cout<<"C("<<x<<") ";}

~C(){cout<<"~C ";}

};

int main()

{

C obiect\_c;

cout<<endl;

B obiect\_b;

cout<<endl;

return 0;

}

**Compileaza :**

A B C

A B

~B ~A ~C ~B ~A

**Explicatie:**

Se creeaza prima data obiectul obiect\_c de tip C. Clasa C mosteneste clasa B, iar clasa B mosteneste clasa A. Astfel, pentru a crea un obiect de tip C, trebuie mai intai apelam constructorul clasei B. Dar pentru asta, trebuie mai intai sa apelam constructorul clasei A.

De aici afisarea de pe prima linie A B C ( ganditi-va la un arbore genealogic bunici-parinti-copii)

Apoi se creaza un obiect de tip B in aceeasi maniera.

Obiectele sunt distruse in oridinea crearii lor . De aici si afisarea de pe ultima linie (Practic este afisarea de la constructori in ordine inversa)

**Exercitiul 2**

**#include<iostream>**

**using namespace std;**

**class A**

**{**

**private:**

**int x;**

**public:**

**A(){ x = 6;}**

**void show\_x () {cout<<x<<' ';}**

**};**

**class B: A**

**{**

**private:**

**int y;**

**public:**

**B(){y=7;}**

**void show\_y() {cout<<y<<' ';}**

**};**

**int main()**

**{**

**B b;**

**b.show\_x();**

**b.show\_y();**

**return 0;**

**}**

**Nu compileaza:**

Daca nu punem explicit un modificator de acces la mostenire, atunci este by default **private.**

**Deoarece clasa B mosteneste clasa A cu modificatorul de acces private, atunci datele publice din A devin private in B. In acest sens, functia show\_x este private in clasa B. Deci, nu va putea fi accesata in afara acestei clase.**

Atunci cand vrem sa apelam metoda in main primim eroarea:

error: 'A' is not an accessible base of 'B'

26| b.show\_x();

| ~~~~~~~~^~

**Modificare** : schimbam modificatorul de acces din private in public : **class B: public A**

**Exercitiul 3**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**public:**

**void f() {cout<<"f din baza"<<endl;}**

**};**

**class B : public A**

**{**

**public:**

**void f() {cout<<"f din derivata"<<endl;}**

**};**

**int main()**

**{**

**B b;**

**b.f();**

**return 0;**

**}**

**Compileaza: f din derivata**

**Exercitiul 4**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**protected:**

**int a;**

**public:**

**B() { a = 7; }**

**};**

**class D : public B**

**{**

**public:**

**int b;**

**D() { b = a + 7; }**

**};**

**int main()**

**{**

**D d;**

**cout << d.b;**

**return 0;**

**}**

**CORECT: se afiseaza 14**

**Explicatie: in clasa B, a e protected, deci D are acces la variabila a, iar b e public**

**Exercitiul 5**

**#include <iostream>**

**using namespace std;**

**class cls1**

**{**

**protected:**

**int x;**

**public:**

**cls1(int i = 10) { x = i; }**

**int get\_x() { return x; }**

**};**

**class cls2 : cls1**

**{**

**public:**

**cls2(int i) : cls1(i) {}**

**};**

**int main()**

**{**

**cls2 d(37);**

**cout << d.get\_x();**

**return 0;**

**}**

**NU COMPILEAZA: cls2 mosteneste privat cls1, deci functia get\_x nu e disponibila in main prin intermediul obiectului d, care e de tip cls2**

**Solutie: modificam mostenirea sa devina public**

**Exercitiul 6**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**int x;**

**public:**

**B(int i = 0) { x = i; }**

**};**

**class D : public B**

**{**

**public:**

**D() : B(15) {}**

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

**};**

**int main()**

**{**

**D d;**

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

**return 0;**

**}**

**NU COMPILEAZA: x este private in B, adica nu poate fi accesat de functia f() din D**

**Solutie: modificam accesul lui x din clasa de baza la protected**

**Exercitiul 6**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**protected:**

**int x;**

**public:**

**A(int i = 14) { x = i; }**

**};**

**class B : A**

**{**

**public:**

**B() : A(2) {}**

**B(const B &b) { x = b.x - 14; }**

**void afisare() { cout << x; }**

**};**

**int main()**

**{**

**B b1, b2(b1);**

**b2.afisare();**

**return 0;**

**}**

**COMPILEAZA: afiseaza -12**

**Explicatie: b1 apeleaza constructorul lui A cu valoarea 2, dupa care b2 apeleaza constructorul de copiere si x = 2 - 14**

**Exercitiul 7**

**#include <iostream>**

**using namespace std;**

**class Base1**

**{**

**public:**

**Base1()**

**{**

**cout << " Base1" << endl;**

**}**

**};**

**class Base2**

**{**

**public:**

**Base2()**

**{**

**cout << "Base2" << endl;**

**}**

**};**

**class Derived : public Base2, public Base1**

**{**

**public:**

**Derived()**

**{**

**cout << "Derived" << endl;**

**}**

**};**

**int main()**

**{**

**Derived d;**

**return 0;**

**}**

**COMPILEAZA: afiseaza**

**Base2**

**Base1**

**Derived**

**Exercitiul 8**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**protected:**

**int x;**

**public:**

**void afisare()**

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

**};**

**class D**

**{**

**public:**

**void afisare()**

**{ cout << "D" << x; }**

**};**

**int main()**

**{**

**D d;**

**d.afisare();**

**return 0;**

**}**

**NU COMPILEAZA: clasa D nu mosteneste deloc clasa B => nu are acces la x**

**Modificare: clasa D mosteneste public clasa B (class D: public class B)**

**Partea 2 - Exercitii medii**

**Exercitiul 1**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**public:**

**void f(int x) {cout<<"f din baza" <<endl;}**

**};**

**class B : public A**

**{**

**public:**

**void f() {cout<<"f din derivata"<<endl;}**

**};**

**int main()**

**{**

**B b;**

**b.f();**

**b.f(3);**

**return 0;**

**}**

**Nu compileaza:**

***IMPORTANT:*  la redefinirea unei funcţii din clasa de baza, toate celelalte versiuni sunt automat ascunse.** Astfel, in derivata (adica in B) versiunea functiei f cu antentul void f(int x) nu va mai exista.

**Modificare:**

**Varianta 1: adaugam functia void f(int x) in corpul clasei B . Astfel se va putea apela in main b.f(3);**

**Varianta 2: stergem parametru din apelul b.f(3) . Adica apelam b.f(). Astfel se va putea apela in main b.f() ( versiunea din derivata).**

**Exercitiul 2**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**protected: int x;**

**public: A(int i = 14) { x = i; }**

**};**

**class B : A**

**{**

**public:**

**B() : A(2) {}**

**B(B& b) { x = b.x - 14; }**

**void afisare() { cout << x; }**

**};**

**int main()**

**{**

**B b1, b2(b1);**

**b2.afisare();**

**return 0;**

**}**

**Compileaza:**  -12

**Explicatie:**

Se creaza obiectul b1 de tip B. Se apeleaza constructorul clasei B ce apeleaza constructorul clasei A ⇒ x = 2 ( cu toate ca avem parametru default in constructorul clasei A, i din antet va lua valoarea pe care am transmis-o ca parametru in A(2) , adica 2). Deci. b1.x=2

Apoi se creaza obiectul de tip b2 de tip B (se apelaza constructorul de copiere al clasei B !!! ). Deci b2.x = b1.x - 14 = 2 -14 = -12

Apelam metoda afisare, iar rezultatul va fi -12.

**Exercitiul 3**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**static int x;**

**public:**

**B(){ x++;}**

**static int get\_x() { return x; }**

**};**

**int B::x;**

**class D : public B**

**{**

**public:**

**D() { x++; }**

**};**

**int main()**

**{**

**D d;**

**d.get\_x();**

**return 0;**

**}**

**Nu compileaza:**

**Variabila statica x este private in B si asa va ramane si in derivata D. Deci nu avem acces la aceasta.**

**Modificare : schimbam modificatorul de acces al variabilei clasei B. In loc de private, il punem protected. Astfel, derivata va avea acces la x.**

**Exercitiul 4**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**protected:**

**static int x;**

**public:**

**B() { x++; }**

**~B() { x--; }**

**static int get\_x() { return x; }**

**};**

**int B::x;**

**class D : public B**

**{**

**public:**

**D() { x++; }**

**~D() { x--; }**

**};**

**int main()**

**{**

**D\* p = new D;**

**cout<<p->get\_x()<< ' ';**

**delete p;**

**cout<<p->get\_x()<< ' ';**

**p = new D;**

**cout << D::get\_x();**

**return 0;**

**}**

**Compileaza : 2 0 2**

**x=0**

**D \*p = new D ⇒ se apeleaza B() ⇒ x=1 , apoi se apeleaza D() ⇒ x=2; si afisam**

**Delete p ⇒ se apeleaza D() ⇒ x=1 , apoi se apeleaza B() ⇒ x=0 si afisam ( De ce D() apoi B() ? obiectele sunt distruste in ordinea crearii lor)**

**p=new D ⇒ se apeleaza B() ⇒ x=1 , apoi se apeleaza D() ⇒ x=2; si afisam**

**X paraseste progrmul cu valoarea 0 . Putem face apelul** cout << D::get\_x();

**pentru ca functia get\_x e statica**

**Exercitiul 5**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**protected:**

**int x;**

**public:**

**B(int x=3) { this->x = x;}**

**B(const B&b) {this->x = b.x; cout<<"B ";}**

**};**

**class D : public B**

**{**

**public:**

**D(int x=2):B(x){}**

**D(const D&) { cout<<"D ";}**

**};**

**int main()**

**{**

**D d1(5);**

**D d2(d1);**

**return 0;**

**}**

**Compileaza: D**

**Pentru ca am redefinit constructorul de copiere al clasei D, acestuia ii revine in totalitate sarcina atribuirii. Astfel, cel al clasei B, nu se mai apeleaza.**

**Daca nu am fi redefinit constructorul de copiere al clasei D si doar pe cel al clasei B, ce s-ar fi afisat?**

**Exercitiul 6**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**public:**

**void afisare()**

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

**};**

**class A: private B**

**{**

**public:**

**int x=0;**

**void afisare()**

**{ cout << "A"; }**

**};**

**class C: public A**

**{**

**public:**

**void afisare()**

**{ cout << "C" << x << " "; }**

**};**

**int main()**

**{**

**C c;**

**c.afisare();**

**return 0;**

**}**

**COMPILEAZA: se afiseaza C0**

**Exercitiul 7**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**protected:**

**int x;**

**public:**

**B(): x(0) {}**

**};**

**class D: public B**

**{**

**public:**

**D(int val) { x = val; }**

**friend ostream& operator<<(ostream& os, D& obj) {**

**obj.x++;**

**os << (++obj.x);**

**return os;**

**}**

**};**

**int main()**

**{**

**D obj(10);**

**cout << obj << endl;**

**return 0;**

**}**

**COMPILEAZA: se afiseaza 12**

**Exercitiul 8**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**protected:**

**int x;**

**public:**

**B(): x(0) {}**

**void citire() {**

**cin >> x;**

**}**

**};**

**class D: public B**

**{**

**public:**

**D(int val) { x = val; }**

**friend ostream& operator<<(ostream& os, D& obj) {**

**obj.citire();**

**os << obj.x;**

**return os;**

**}**

**};**

**int main()**

**{**

**D obj;**

**cout << obj << endl;**

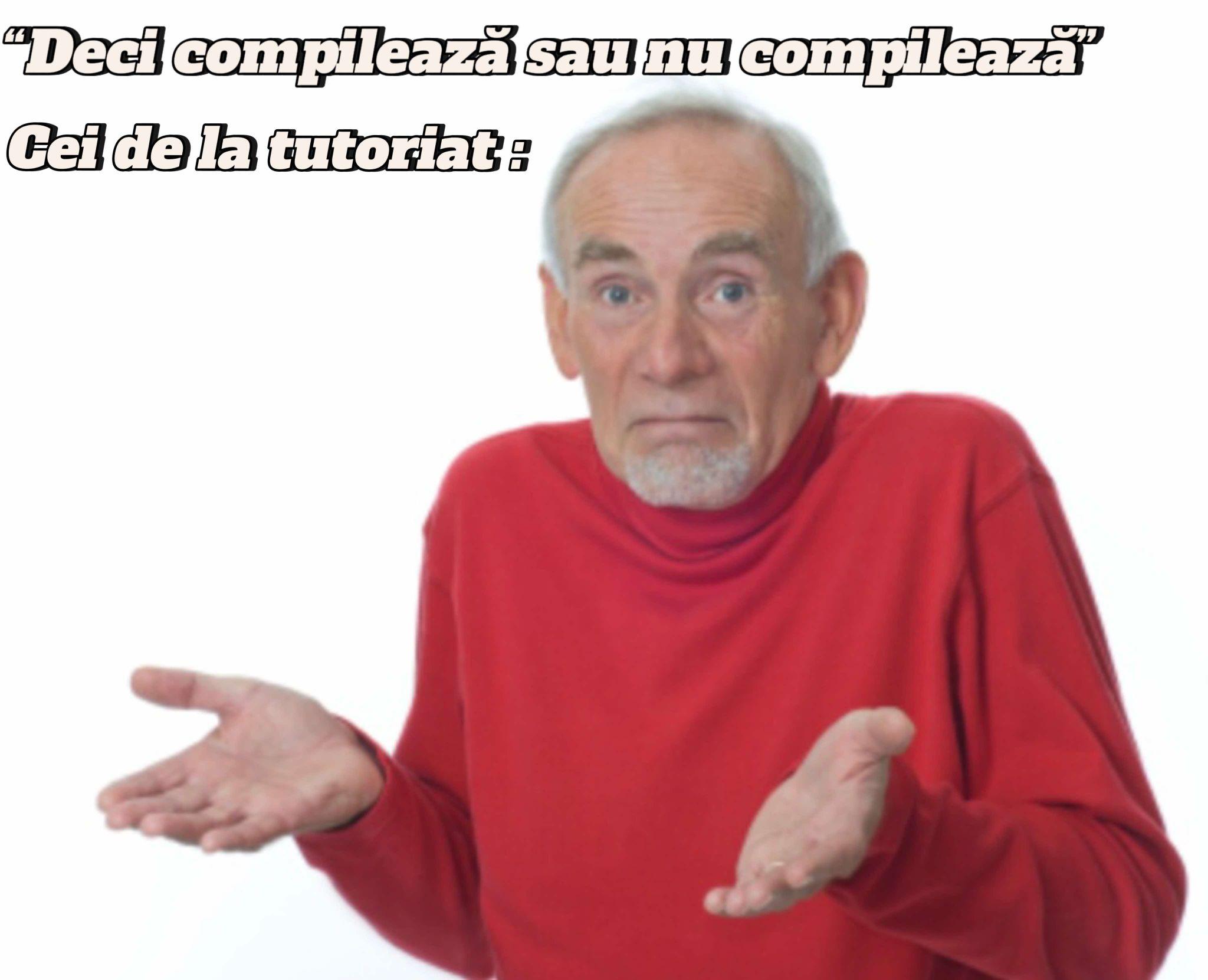
**return 0;**

**}**

**NU COMPILEAZA: obj foloseste constructorul default care nu mai exista**

**Solutie: modificam in D(int val=valoare\_standard) { x=val; }**

**Partea 3 - Exercitii mai grele**

****

**Exercitiul 1**

#include <iostream>

using namespace std;

class A

{

protected: int x;

public: A(int i):x(i){}

int get\_x(){ return x; }

};

class B: A

{

protected: int y;

public: B(int i,int j):y(i),A(j){}

int get\_y(){ return get\_x()+y; }

};

class C: protected B

{ protected: int z;

public: C(int i,int j,int k):z(i),B(j,k){}

int get\_z(){ return get\_x()+get\_y()+z; }

};

int main()

{

C c(5,6,7);

cout<<c.get\_z();

return 0;

}

**Nu compileaza:**

error: 'A' is not an accessible base of 'C'

20 | int get\_z(){ return get\_x()+get\_y()+z; }

Metoda get\_x nu este accesibila in clasa C

Clasa B mosteneste clasa A private, astfel toti membri public din A devin private in B ⇒ get\_x private in B.

Clasa C mosteneste clasa B protected, astfel toti membri public din clasa B devin protected in C. Dar get\_x e private in B, deci clasa C nu are acces la aceasta metoda get\_x;

**Modificare:** schimbam modificatorul de acces la mostenirea clasei B : class B: public A

Obs: era in regula daca punem si modificatorul de acces protected

**Exercitiul 2**

#include<iostream>

using namespace std;

class B

{

protected: int i;

public:

B(int j=5) {cout << " cb "; i=j-2; }

~B(){ cout << " db ";}

int get\_i() { return i; }

};

class D1: public B

{

int j;

public:

D1(int k=10) {cout << " cd1 "; j=i-k+3; }

~D1(){ cout << " dd1 ";}

};

class D2: public D1

{

int k;

public:

D2(int l=15) {cout << " cd2 "; k=i-l+3; }

~D2(){ cout << " dd2 ";}

};

D1 f(D1 d1, D2 d2) {return d1.get\_i()+d2.get\_i(); }

int main()

{

D2 ob2; D1 ob1(3);

cout<<f(ob1,ob2).get\_i()+ob2.get\_i();

return 0;

}

**Compileaza:**

**cb cd1 cd2 cb cd1 cb cd1 6 dd1 db dd1 db dd2 dd1 db dd1 db dd2 dd1 db**

**Exercitiul 3**

#include <iostream>

using namespace std;

class A {

public:

int x;

A() { x = 10; }

};

class B {

public:

int x;

B() { x = 20; }

};

class C : public A, public B {};

int main()

{

C obj;

cout<<obj.x<<' ';

return 0;

}

**Nu compileaza:**

**Avem ambiguitate : clasa C mosteneste clasa A si clasa B iar de la fiecare dintre acestea variabila x. Atunci cand se cere afisarea sa, nu stie la care se refera : x din A sau x din B?**

**Modificare: clasa C va mosteni doar o singura clasa ( A sau B )**

**Acest exemplu ilustreaza mostenirea multipla: adica o clasa poate sa mosteneasca mai multe clase**

**Exercitiul 4**

#include <iostream>

using namespace std;

class B

{

int x;

public:

B(int i = 2): x(i){}

int get\_x() const { return x; }

};

class D : public B

{

int\* y;

public:

D(int i = 2): B(i)

{

y = new int[i];

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

y[j] = 1;

}

D(D& a)

{

y = new int[a.get\_x()];

for (int i = 0; i < a.get\_x(); i++)

y[i] = a[i];

}

int& operator[](int i) { return y[i]; }

};

ostream& operator<<(ostream& o, const D& a)

{

for (int i = 0; i < a.get\_x(); i++)

o << a[i];

return o;

}

int main()

{

D ob(5);

cout << ob;

return 0;

}

***Din tutoriatele trecute*…..Nu compileaza: obiect constant apeleaza metoda neconstanta**

**Aici avem un obiect constant : const D&a:**

ostream& operator<<(ostream& o, const D& a)

{

for (int i = 0; i < a.get\_x(); i++)

o << a[i];

return o;

}

**El incearca sa apeleze o metoda neconstanta: int&operator[] (int i)**

int& operator[](int i) { return y[i]; }

**Modificare:**

**Varianta 1: Facem obiectul a neconstant ( ceea ce nu e neaparat recomandat)**

**Varianta 2: Facem metoda constanta : int&operator[] (int i) const**

**Exercitiul 5**

**#include <iostream>**

**using namespace std;**

**class Baza**

**{**

**protected:**

**int x;**

**public:**

**Baza(int i) { x = i; }**

**};**

**class Derivata : public Baza**

**{**

**public:**

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

**void print() { cout << x; }**

**};**

**int main()**

**{**

**Derivata d(10);**

**d.print();**

**}**

**NU COMPILEAZA: constructorul din Derivata nu poate initializa membrii bazei in lista de initializare, ci doar printr-un constructor al clasei de baza sau in corpul constructorului**

**Modificare: Derivata(int i): Baza(i) {}**

**Exercitiul 6**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**protected:**

**int x;**

**public:**

**B(): x(0) {}**

**friend istream& operator>>(istream& is, B& obj) {**

**int n;**

**is >> n;**

**obj.x = n++;**

**return is;**

**}**

**};**

**class D: public B**

**{**

**public:**

**D(int val) { x = val; }**

**friend ostream& operator<<(ostream& os, const D& obj) {**

**os << obj.x;**

**return os;**

**}**

**};**

**int main()**

**{**

**D obj(10);**

**cin >> obj;**

**cout << obj << endl;**

**return 0;**

**}**

**COMPILEAZA: afiseaza valoarea introdusa de la tastatura.**

**Explicatie: se face conversie implicita de la D la B la apelul citirii deoarece nu a fost adaugata nicio variabila in D**

**Exercitiul 7**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**protected: static int x;**

**private: int y;**

**public: A(int i) { x=i; y=-i+4; }**

**int put\_x(A a) { return a.x+a.y; }**

**};**

**int A::x=7;**

**int main()**

**{**

**A a(10);**

**cout<<a.put\_x(20);**

**return 0;**

**}**

**COMPILEAZA: se afiseaza 4**

**Explicatie: a(10) => x=10, y=-6**

**a.put\_x(20): 20 se converteste la clasa A => x devine 20, y=-16**

**acum a.x este 20, a.y este -16 => suma lor e 4**

**Exercitiul 8**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**public:**

**int x;**

**B() : x(1) {}**

**B(B& obj) : x(obj.x) {}**

**};**

**class D: public B**

**{**

**int y;**

**public:**

**D(): y(2) {}**

**D(B obj)**

**{**

**y = obj.x;**

**}**

**friend ostream& operator<<(ostream& os, const D& obj)**

**{**

**os << (++obj.x) << " " << obj.y;**

**return os;**

**}**

**};**

**int main()**

**{**

**B a;**

**D b(a);**

**cout << b;**

**}**

**NU COMPILEAZA: operatorul << incearca sa modifice obiectul obj**

**Solutie: fie scoatem const, fie scoatem ++ de la obj.x din operatorul <<**