**Exercitii Tutoriat 5**

→ Polimorfism, Virtualizare

**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 I: Exercitii Usoare**

**Exercitiul 1**

**#include <iostream>**

**using namespace std;**

**class Baza**

**{**

**public:**

**int x;**

**Baza(): x(10) {}**

**void afisare() // functia din baza**

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

**};**

**class Derivata : public Baza**

**{**

**public:**

**Derivata(): x(5) {}**

**void afisare() // suprascrierea functiei din baza**

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

**};**

**int main()**

**{**

**Baza \*p = new Derivata;**

**p->afisare();**

**}**

EROARE: constructorul nu poate initializa variabila x in lista de initializare

Modificare: Derivata(): Baza()

**Exercitiul 2**

**#include <iostream>**

**using namespace std;**

**class Baza**

**{**

**public:**

**int x;**

**Baza(int i=10): x(i) {}**

**void afisare() // functia din baza**

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

**};**

**class Derivata : public Baza**

**{**

**public:**

**Derivata(): Baza(5) {}**

**void afisare() // suprascrierea functiei din baza**

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

**};**

**int main()**

**{**

**Baza \*p = new Derivata;**

**p->afisare();**

**}**

COMPILEAZA: se afiseaza Baza5

**Exercitiul 3**

#include <iostream>

using namespace std;

class Derivata

{

public:

int x;

Derivata(int i=10): x(i) {}

void afisare() // functia din baza

{ cout << "Baza"<<x; }

};

class Baza: public Derivata

{

public:

Baza(): Derivata(5) {}

void afisare() // suprascrierea functiei din baza

{ cout << "Derivata" << x; }

};

int main()

{

Baza \*p = new Derivata;

p->afisare();

}

NU COMPILEAZA: Baza mosteneste Derivata, nu putem scrie Baza \*p = new Derivata.

Modificare: Derivata \*p = new Baza;

**Exercitiul 4**

#include <iostream>

using namespace std;

class Baza

{

public:

int x;

Baza(int i=10): x(i) {}

void afisare() // functia din baza

{ cout << "Baza"<<x; }

};

class Derivata : public Baza

{

public:

Derivata(): Baza(5) {}

void afisare() // suprascrierea functiei din baza

{ cout << "Derivata" << x; }

};

int main()

{

Derivata \*p = new Baza;

p->afisare();

}

NU COMPILEAZA: un pointer de tip Derivata nu poate stoca un obiect de tip Baza

Modificare: Derivata \*p = new Derivata

SAU

Baza \*p = new Derivata

**Exercitiul 5**

#include <iostream>

using namespace std;

class Baza

{

public:

int x;

Baza(int i=10): x(i) {}

void afisare() // functia din baza

{ cout << "Baza"<<x; }

};

class Derivata : public Baza

{

public:

Derivata(int x=100): Baza(5) {}

void afisare() // suprascrierea functiei din baza

{ cout << "Derivata" << x; }

};

int main()

{

Derivata \*p = new Derivata(60);

p->afisare();

}

COMPILEAZA: afiseaza Derivata5

**Exercițiul 6 – Vezi fisierul Redefinirea operatorilor - virtual**

**Partea II: Exercitii Medii**

**Exercitiul 1**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**public:**

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

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

**};**

**class B : public A**

**{**

**public:**

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

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

**};**

**class C : public B {**

**public:**

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

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

**};**

**int main()**

**{**

**C \*c = new C;**

**B ob = \*c;**

**return 0;**

**}**

**Compileaza: ABC~B~A**

**Explicatie:**

**pentru C\*c =new C se afiseaza ABC ( nu se va apela niciun destructor in final pentru aceasta parte). De ce nu se afiseaza si ~C~B~A. Pentru ca noi am alocat memoria, deci noi trebuie sa o dezalocam cu delete c (Abia atunci s-ar fi afisat si ~C~B~A).**

**B ob =\*c se afiseaza ~B~A. De ce nu se afiseaza si A B? Pentru ca in acest caz se apeleaza Copy Contructor-ul, nu constructorul Default!!!**

**Exercitiul 2**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**int i;**

**public: B() { i=80; }**

**virtual int get\_i() { return i; }**

**};**

**class D: public B**

**{**

**int j;**

**public: D() { j=27; }**

**int get\_j() {return j; }**

**};**

**int main()**

**{**

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

**cout<<p->get\_i();**

**cout<<((D\*)p)->get\_j();**

**return 0;**

**}**

**Nu compileaza: invalid conversion from 'B\*' to 'D\*' [-fpermissive] D \*p=new B;**

**Explicatie:**

**un obiect de tip derivata nu poate pointa catre baza fara a folosi un mecanism special.**

**Exercitiul 3**

**#include<iostream>**

**using namespace std;**

**class A**

**{**

**public:**

**int x;**

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

**virtual A minus() { return (1 - x); }**

**};**

**class B : public A**

**{**

**public:**

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

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

**};**

**int main()**

**{**

**A \*p1 = new B(18);**

**\*p1 = p1->minus();**

**p1->afisare();**

**return 0;**

**}**

**Nu compileaza: 'class A' has no member named 'afisare' p1->afisare();**

**Explicatie: Eroarea apare la apelul functiei afisare. Avem un obiect de tip baza ce pointeaza catre derivata. Astfel, prima data se va cauta functia afisare in baza si se va constata ca acesta nu exista. La apelul functiei minus nu avem nicio problema.**

**Exercitiul 4**

**#include<iostream>**

**using namespace std;**

**class A**

**{ int i;**

**public: A(int x=2):i(x+1) {}**

**virtual int get\_i() { return i; }};**

**class B: public A**

**{ int j;**

**public: B(int x=20):j(x-2) {}**

**virtual int get\_j() {return A::get\_i()+j; }};**

**int main()**

**{**

**A o1(5);**

**B o2;**

**cout<<o1.get\_i()<<" ";**

**cout<<o2.get\_j()<<" ";**

**cout<<o2.get\_i();**

**return 0;**

**}**

**COMPILEAZA: afiseaza 6 21 3**

**Exercitiul 5**

**#include<iostream>**

**using namespace std;**

**class B**

**{**

**int\*x;**

**public:**

**B(){this->x=new int(0);}**

**virtual ~B(){cout<<"~B "; delete x;}**

**};**

**class D : public B**

**{**

**int\*y;**

**public:**

**D(){this->y=new int(3);}**

**~D(){cout<<"~D ";delete y;}**

**};**

**class F : public D**

**{**

**int\*z;**

**public:**

**F(){this->z=new int(7);}**

**~F(){cout<<"~F "; delete z;}**

**};**

**int main()**

**{**

**B\*b =new F;**

**return 0;**

**}**

**Compileaza: Nu afiseaza nimic**

**Exercitiul 6**

**#include<iostream>**

**using namespace std;**

**class A**

**{ int i;**

**public: A(int x=10):i(x) {}**

**int get\_i() { return i; }};**

**class B: public A**

**{**

**public: B(int x=20) {}**

**virtual int get\_i() {return A::get\_i()+1; }};**

**class C: public B**

**{**

**public: C() {}**

**int get\_i() {return A::get\_i()+5; }};**

**int main()**

**{**

**A \*p = new C;**

**B \*q = new C;**

**B \*r = new B;**

**cout<<q->get\_i()<<" "<<r->get\_i()<<" "<<p->get\_i();**

**return 0;**

**}**

**COMPILEAZA: afiseaza 15 11 10**

**Partea III: Exercitii Mai Grele**

**Exercitiul 1**

**#include<iostream>**

**using namespace std;**

**class B**

**{**

**public:**

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

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

**};**

**class D:B**

**{**

**public:**

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

**};**

**int main()**

**{**

**B\*b=new D();**

**b->f();**

**return 0;**

**}**

**Nu compileaza: error: 'B' is an inaccessible base of 'D'**

**Explicatie:**

**Clasa D are acces la toti membrii publici si protected ai clasei B.Dar codul exterior (in afara clasei D) nu poate trata D ca un obiect de tip B.**

**Din punct de vedere al accesibilitatii: Daca mostenirea este privata, partea B a obiectului D este ascunsa in afara clasei**

**Modificare: B\*b=new B(); sau mostenire publica**

**Exercitiul 2**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**protected:**

**int x;**

**public:**

**B(int i = 28) { x = i; }**

**virtual B f(B ob) { return x + ob.x + 1; }**

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

**};**

**class D : public B**

**{**

**public:**

**D(int i = -32): B(i) {}**

**B f(B ob) { return x + ob.x - 1; }**

**};**

**int main()**

**{**

**B \*p1 = new D, \*p2 = new B, \*p3 = new B(p1->f(\*p2));**

**p3->afisare();**

**return 0;**

**}**

**Nu compileaza:**

**'int B::x' is protected within this context ( se refera la B f(b ob) din clasa D)**

**16 | B f(B ob) { return x + ob.x - 1; }**

**| ^**

**Explicatie: Clasa D mosteneste public clasa B ⇒ x ramane protected. Ceea ce inseamna ca x poate fi accesat dar doar pentru obiecte ale clasei curente sau clase derivate.**

**Deci, membrul x este accesibil în clasa D, dar doar pentru instante de tip D sau B mostenite de D. Nu este permis accesul la x dintr-un alt obiect de tip B, cum este ob în aceasta situatie.**

**Modificare: B f(D ob) { return x + ob.x - 1; } → argumentul devine de tip D**

**Exercitiul 3**

**#include <iostream>**

**using namespace std;**

**class A {**

**public:**

**virtual int f() const{ cout << "A::f()\n"; return 1;}**

**virtual void f(string) const { cout << "A::f(string)\n"; }**

**virtual void g() const { cout << "A::g()\n"; }**

**};**

**class B : public A {**

**public:**

**void g() const { cout << "B::g()\n"; }**

**};**

**class C : public A {**

**public:**

**int f() { cout << "C::f()\n"; return 2; }**

**};**

**int main()**

**{**

**string s("Ce cauti straine la clasa mea...");**

**B ob1;**

**ob1.g();**

**ob1.f(s);**

**const C ob2;**

**int x = ob2.f();**

**return 0;**

**}**

**Nu compileaza:**

**Obiectul constant ob2 apleaza o functie neconstanta f.**

**Exercitiul 4**

**#include <iostream>**

**using namespace std;**

**class Baza**

**{**

**public:**

**virtual static void afisare() { cout << "Baza "; }**

**virtual void afisare2() const {cout<<"Baza2 ";}**

**};**

**class Derivata : public Baza**

**{**

**public:**

**void afisare() { cout << "Derivata "; }**

**};**

**int main()**

**{**

**Derivata \*d = new Derivata;**

**const Baza\* b= new Derivata;**

**d->afisare();**

**b->afisare2();**

**return 0;**

**}**

**Nu compileaza: nu putem avea metode virtuale statice**

**Exercitiul 5**

**#include <iostream>**

**using namespace std;**

**class B {**

**protected:**

**int x;**

**public:**

**B(int i = 12) { x = i; }**

**virtual B f(B ob) { return x + ob.x + 1; }**

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

**};**

**class D : public B {**

**public:**

**D(int i = -15)**

**: B(i - 1)**

**{**

**x++;**

**}**

**B f(B ob) { return x - 2; }**

**};**

**int main()**

**{**

**B \*p1 = new D, \*p2 = new B, \*p3 = new B(p1->f(\*p2));**

**p3->afisare();**

**return 0;**

**}**

**COMPILEAZA: se afiseaza -17**

**Explicatii: in initializarea lui p3, cu o baza, se intra pe functia f din p1 cu valoarea lui p2 ca parametru.**

**Deci: pentru p1, x=-15. Pentru p2, x=12. Pentru p3, x= (x-ul de la p1) - 2 =-17**

**EXERCITIUL 6**

**#include <iostream>**

**using namespace std;**

**class B {**

**int i;**

**public:**

**B() { i = 1; }**

**virtual int get\_i() { return i; }**

**};**

**class D : public B {**

**int j;**

**public:**

**D() { j = 2; }**

**int get\_i() { return B::get\_i() + j; }**

**};**

**int main()**

**{**

**const int i = cin.get();**

**if (i % 3) {**

**D o;**

**}**

**else {**

**B o;**

**}**

**cout << o.get\_i();**

**return 0;**

**}**

**NU COMPILEAZA: incercam sa afisam ceva din obiectul 0, dar acesta nu exista in afara if-else-ului**

**Modificare: declaram un “B o” sau un “D o” inainte de linia de afisare**