**Exercitii Tutoriat 2**

→ constructori & destructori

**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**

**{**

**private:**

**int x;**

**public:**

**void setX (int x) { Test::x = x; }**

**void print() { cout << "x = " << x << endl; }**

**};**

**int main()**

**{**

**Test obj;**

**int x = 40;**

**obj.setX(x);**

**obj.print();**

**return 0;**

**}**

**Compileaza : afiseaza x=40**

**Explicatie: in functia setX se stie precis ca vrem sa initializam variabila x, data membra a clasei Test. De ce se stie la care x ne referim? Pentru ca folosim operatorul de rezolutie ::**

**Cu alte cuvinte, spunem ca x data membra a clasei (Test::x) va avea valoarea parametrului x din antetul functiei.**

**La fel de bine se putea folosi si pointerul this : this->x = x;**

**Exercitul 2:**

**#include<iostream>**

**using namespace std;**

**class Test**

**{**

**int &t;**

**public:**

**Test (int &x) { t = x; }**

**int getT() { return t; }**

**};**

**int main()**

**{**

**int x = 20;**

**Test t1(x);**

**cout << t1.getT() << " ";**

**x = 30;**

**cout << t1.getT() << endl;**

**return 0;**

**}**

**Nu compileaza :**

**Referinta nu este initializata : int&t;**

**(e foarte important sa intelegeti mesajele de eroare din consola)**

**"Test::Test(int &x)" provides no initializer for: reference member "Test::t"**

**Modificare:**

**Varinata1:**

**Stergem referinta de la t. Adica t va fi de tip int, nu int& → int t;**

**Varianta2:**

**Folosim lista de initializare a constructorilor**

**Lista de initializare a constructorilor este o sectiune speciala a constructorului folosita pentru a initializa direct membri clasei (inainte de executarea corpului constructorului).**

**Astfel, daca initializam referinta int& t cu ajutor listei de initializare a constructorilor, atunci nu vom obtine eroare la compilare.**

**class Test**

**{**

**int& t;**

**public:**

**Test (int &x): t(x) { }**

**int getT() { return t; }**

**};**

**Cand o referinta este un membru al unei clase, aceasta trebuie initializata în lista de initializare a constructorului. Daca încercam sa o initializam in corpul constructorului (cum faceam initial), C++ consideră ca incercam sa „reasignam” referina, lucru nepermis.**

**Exercitiul 3**

**#include<iostream>**

**using namespace std;**

**class Test**

**{**

**int value;**

**public:**

**Test(int v = 13);**

**};**

**Test::Test(int v)**

**{**

**value = v;**

**cout<<value<<' ';**

**}**

**int main()**

**{**

**Test t[10];**

**return 0;**

**}**

**Compileaza: 13 13 13 13 13 13 13 13 13 13 ( 13 de 10 ori adica)**

**Explicatie: se apeleaza constructorul default pentru fiecare element al array-ului t de tip Test. Nu obtinem eroare la compilare daca la definirea corpului constructorului in afara clasei nu trecem valoarea parametrilor default (adica daca nu punem v=13)**

**Exercitiul 4**

**#include <iostream>**

**using namespace std;**

**class Test**

**{**

**public:**

**Test(const Test& t)**

**{**

**cout << "Copy " << endl;**

**}**

**Test& operator=(const Test& t)**

**{**

**cout << "Egal " << endl;**

**return \*this;**

**}**

**};**

**int main()**

**{**

**Test t1, t2;**

**t2 = t1;**

**Test t3 = t1;**

**return 0;**

**}**

**Nu compileaza: Nu avem constructorul fara parametri**

**Explicatie:**

**-t1 si t2 au nevoie sa apeleze constructorul fara parametri**

**-pentru ca am definit explicit constructorul de copiere, compilatorul nu va mai genera automat un constructor default.**

**Modificare: adaugam constructorul fara parametri Test();**

**IMPORTANT: De fiecare data cand definim unul sau mai multi constructori non-impliciti (cu parametri, copy constructor ) pentru o clasa, un constructor implicit (fara parametri) ar trebui de asemenea sa fie definit in mod explicit, deoarece compilatorul nu va furniza un constructor implicit in acest caz. Totusi, nu este necesar mereu, dar este considerat o buna practica sa definim intotdeauna un constructor implicit.**

**Exercitul 5**

**#include <iostream>**

**using namespace std;**

**class Test**

**{**

**public:**

**Test()**

**{**

**cout<<"Default "<<endl;**

**}**

**Test(const Test& t)**

**{**

**cout << "Copy " << endl;**

**}**

**Test& operator=(const Test& t)**

**{**

**cout << "Egal " << endl;**

**return \*this;**

**}**

**};**

**int main()**

**{**

**Test t1, t2;**

**t2 = t1;**

**Test t3 = t1;**

**return 0;**

**}**

**Compileaza:**

**Default**

**Default**

**Egal**

**Copy**

**Explicatie:**

**Primul Default pentru t1**

**Al doilea Default pentru t2**

**Egal pentru t2= t1**

**Copy pentru t3 (t3 este initializat in momentul declararii)**

**Exercitul 6**

**#include <iostream>**

**using namespace std;**

**class Test**

**{**

**int id;**

**public:**

**Test(int x) { id = x; }**

**void display() { cout<<id<<’ ‘; }**

**};**

**int main()**

**{**

**Test obj1(10);**

**obj1.display();**

**Test obj2(obj1);**

**obj2.display();**

**return 0;**

**}**

**Compileaza : 10 10**

**Explicatie:**

**Se apeleaza constructorul de copiere, cel dat de compilator**

**Exercitul 7**

**#include<iostream>**

**using namespace std;**

**class B**

**{**

**int x;**

**public:**

**B() {x=2; cout<<"B";}**

**B(int x) {B::x=x; cout<< "B("<<x<<")";}**

**};**

**class C**

**{**

**B b;**

**public:**

**C() { b = B(2025);}**

**};**

**int main()**

**{**

**C c;**

**return 0;**

**}**

**Compileaza: BB(2025)**

**Explicatie:**

**-pornim de la obiectul c. acesta trebuie sa apeleze constructorul sau default**

**-clasa C are un obiect de tip B. deci clasa C nu poate exista daca nu exista clasa B. Asadar, se va apela constructorul default pentru obiectul b de tip B al clasei C. De aici rezulta primul B.**

**-apoi c apelează constructorul default al clasei C. Aici are loc apelul B(2025), de unde rezulta ultima parte a output-ului**

**ATENTIE: { b = B(2025);} aici sunt doua apeluri**

**-o data B(2025) care apeleaza constructorul cu parametri**

**-a doua oara , apelul operatorului =**

**Exercitiul 8**

**#include <iostream>**

**using namespace std;**

**class cls {**

**int x;**

**public:**

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

**int set\_x(int i)**

**{**

**int y = x;**

**x = i;**

**return y;**

**}**

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

**};**

**int main()**

**{**

**cls\* p = new cls[10];**

**int i = 0;**

**for (; i < 10; i++)**

**p[i].set\_x(i);**

**for (i = 0; i < 10; i++)**

**cout << p[i].get\_x();**

**return 0;**

**}**

**Nu compileaza:**

**Nu avem constructor default**

**Exerciul 9**

**#include <iostream>**

**using namespace std;**

**class Test {**

**int x;**

**public:**

**Test(int val) : x(val) {}**

**Test(const Test);**

**void afisare() const**

**{**

**cout << "x: " << x << endl;**

**}**

**};**

**int main() {**

**Test obj1(4);**

**Test obj2 = obj1;**

**obj2.afisare();**

**return 0;**

**}**

**//aici sunt doua greseli de corectat:**

**Nu compileaza:**

**1)Signatura constructorului de copiere este gresita : corect e Test(const Test&);**

**2)Nu are corpul defini**

**Daca nu scriem macar Test(const Test&){} atunci se considera ca functia ar trebui sa fie definita in afara clasei ⇒ eroare run-time. Totusi, pentru a obtine valorile asteptate ( adica nu o valoare random din memorie), de defineste COMPLET cu toate datele**

**adica:**

**#include <iostream>**

**using namespace std;**

**class Test {**

**int x;**

**public:**

**Test(int val) : x(val) {}**

**Test(const Test&y) {x=y.x;}**

**void afisare() const**

**{**

**cout << "x: " << x << endl;**

**}**

**};**

**int main() {**

**Test obj1(15);**

**Test obj2 = obj1;**

**obj2.afisare();**

**return 0;**

**}**

**Exercitiul 10:**

**#include <iostream>**

**using namespace std;**

**class Test**

**{**

**int\* data;**

**public:**

**Test(int val) { data = new int(val);}**

**Test(const Test& obj){data = new int(\*obj.data);}**

**Test& operator=(const Test&obj)**

**{**

**delete data;**

**data = obj.data;**

**return \*this;**

**}**

**void afis() const {cout << "Value: " << \*data << endl;}**

**~Test(){ delete data; }**

**};**

**int main()**

**{**

**Test obj1(10);**

**Test obj2(20);**

**obj2 = obj1;**

**obj2.afis();**

**return 0;**

**}**

**Nu compileaza :Eroarea apare din cauza definirii opeatorului=**

**-delete data, dezaloca memorie pentru variabila data**

**-apoi pointerul data va lua valorea pointer-ului obj.data, ceea ce inseamna ca data în obj2 va pointeriza la aceeasi zona de memorie ca si obj1**

**-apare o problemă de tip double free atunci când ambele obiecte vor fi distruse - ambele vor incerca sa elibereze aceeasi zona de memorie**

**-prima data elibereaza memorie obj2, pentru ca este al doilea obiect creat**

**-cand obj1 incearca se elibereze memoria, nu mai are ce elibera pentru ca deja o facuse obj2.**

**Corect este:**

**Test& operator=(const Test& obj)**

**{**

**if (data!=NULL)**

**delete data; // Eliberare memorie veche**

**data = new int(\*obj.data); // Alocare memorie noua si copierea valorii**

**return \*this;**

**}**

**-mai intai vedem daca variabila data nu este nula (adica daca avem ce dezaloca)**

**-apoi alocam memorie pentru variabila data, ce va retine valoarea DE LA adresa de memorie indicata de pointer-ul obj.data**

**-la final returnam \*this ⇒ in caz contrar, obtinem eroare de compilare**

**Exercitiul 11:**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**private:**

**int x;**

**public:**

**A(): x(3)**

**{ cout<<"C1"<<" "; }**

**A(int x): x(x)**

**{ cout<<"C2"<<" "; }**

**A(const A &a): x(a.x)**

**{ cout<<"C3"<<" "; }**

**const A& operator=(const A &a)**

**{**

**x = a.x;**

**cout<<"A"<<" ";**

**return \*this;**

**}**

**~A()**

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

**};**

**int main()**

**{**

**A a1, a2(5), a3(a2), a4;**

**a4 = a3;**

**return 0;**

**}**

**Compileaza**  
**Afiseaza C1 C2 C3 C1 A D D D D**

**Exercitiul 12:**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**private:**

**int x, y;**

**public:**

**A():x(3), y(5) {}**

**A(int x): x(x), y(6) {}**

**A(int y): x(7), y(y) {}**

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

**void afisare()**

**{**

**cout<<x<<" "<<y<<" ";**

**}**

**~A()**

**{**

**cout<<"D"<<" ";**

**}**

**};**

**int main()**

**{**

**A a, b(12), c(15), d(20, 25);**

**a.afisare();**

**d.afisare();**

**return 0;**

**}**

**Nu compileaza: eroare la constructorul A(int y) -> A::A(int) cannot be overloaded with A::A(int) -> avem deja un constructor cu un singur parametru int**

**Modificare: eliminarea constructorului A(int y)**

**Exercitiul 13:**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**private:**

**int x;**

**public:**

**B(): x(3) { cout<<"C "; }**

**B(int x) { x = x; cout<<"C2 "; }**

**B(const B &obj): x(obj.x) { cout<<"CC ";}**

**const B& operator= (const B &obj)**

**{**

**if (this != &obj)**

**{**

**x = obj.x;**

**}**

**cout << "= ";**

**return \*this;**

**}**

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

**};**

**int main()**

**{**

**B a, b(4), c=a;**

**a=b;**

**return 0;**

**}**

**Compileaza**

**Afiseaza C1 C2 CC = D D D**

**Exercitiul 14:**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**private:**

**int x;**

**int \*y;**

**public:**

**A():x(3)**

**{ y = new int(5); }**

**~A()**

**{ if( y != NULL )**

**delete y; }**

**};**

**class A**

**{**

**private:**

**int x, y;**

**public:**

**A():x(3), y(5) {}**

**A(int x): x(x), y(6) {}**

**A(int x,int y=8): x(x), y(y) {}**

**void afisare()**

**{ cout<<x; }**

**~A() {}**

**};**

**int main()**

**{**

**A a, b(12,10);**

**return 0;**

**}**

**Nu compileaza. Eroare la a doua declarare a clasei A: error: redefinition of ‘class A’**

**Modificare: schimbarea numelui clasei (implicit a constructorilor si destructorului)**

**Exercitiul 15:**

**#include <iostream>**

**using namespace std;**

**class B**

**{**

**private:**

**int x;**

**public:**

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

**B(const B& obj)**

**{**

**x=obj.x;**

**}**

**~B() {}**

**int getX()**

**{**

**return x;**

**}**

**}**

**int main()**

**{**

**B a(8), b(a);**

**cout<< a.getX() << " " <<b.getX();**

**}**

**Nu compileaza. Eroare la finalul clasei: expected ‘;’**

**Modificare: adaugam ;**

**Exercitiul 16:**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**private:**

**int x;**

**int \*y;**

**public:**

**A():x(3)**

**{ y = new int(5); }**

**A(int y):x(4)**

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

**~A()**

**{ if( y != NULL )**

**delete y; }**

**void afisare()**

**{ cout<<x<<" "<<\*y<<" "; }**

**void setY(int y)**

**{ \*this->y = y; }**

**};**

**int main()**

**{s**

**A a, b(6);**

**a=b;**

**b.setY(10);**

**a.afisare();**

**b.afisare();**

**return 0;**

**}**

**Compileaza.**

**Afiseaza 4 10 4 10**