

Nume
Grupa

Nr. 1

14 iunie 2024
Programare orientată pe obiecte

Examen scris

- I. Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ propuneți o (singură) modificare prin care programul devine corect.

```
1. #include<iostream>
2. using namespace std;
3. class Base {
4. public:
5. int f() const { cout << "B1\n"; return 1; }
6. int f(string) const {
7.     cout << "B2\n"; return 1; } };
8. class Derived : public Base {
9. public: int f() const
10.     { cout << "D1\n"; return 2; } };
11. int main() {
12.     string s("hello");
13.     Derived d;
14.     int x = d.f();
15.     d.f(s);
16.     return 0; }
```

Programul compilează? DA ☐ NU ☒

Daca DA ce se afișează pe ecran:

Daca NU: de ce nu?

Redefinirea functiei f() in derivata o ascunde pe f(string)

modificarea care îl face să meargă (o singură linie modificată, precizat nr linie modificată si modificarea)

eliminat linia 15 sau la linia 13 d definit de tip Base, si altele

Barem: 0.3 pentru eroare +
0.2 pentru modificare

- II. Descrieți pe scurt folosirea cuvântului cheie virtual în C++.

Barem: 0.1 mentionat functii virtuale

0.1 mentionat problema rombului la mostenirea multipla

0.2 descris functiile virtuale: 0.1 pointer de tip baza catre obiect derivat

0.1 sintaxa functiei, explicat RTTI

0.1 ca trebuie virtual pe ambele mosteniri ca sa se evite ambiguitatea de la romb

Doar functii virtuale: 0.3

Doar romb 0.2

- III. Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ propuneți o (singură) modificare prin care programul devine corect.

```
1. #include <iostream>
2. using namespace std;
3. class cls {
4. int x;
5. public: cls(int i = 1) { x=i; }
6. cls (): x(-1) { }
7. int set_x(int i) { int y=x; x=i; return y; }
8. int get_x(){ return x; } };
9. int main()
10. { cls *p=new cls[10];
11. for(int i=3;i<9;i++) p[i].set_x(i);
12. for(int i=0;i<4;i++) cout<<p[i].get_x();
13. return 0;
14. }
```

Programul compilează? DA ☐ NU ☒

Daca DA ce se afișează pe ecran:

Daca NU: de ce nu?

Ambiguitate între cei 2 constructori ai clasei cls la crearea vectorului de obiecte modificarea care il face sa mearga (o singura linie modificata, precizat nr linie modificat)

Barem: 0.3 pentru eroare +
0.2 pentru modificare

- IV. Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ propuneți o (singură) modificare prin care programul devine corect.

```
1. #include <iostream>
2. using namespace std;
3. class A {
4.     static int *x;
5.     public: A() {}
6.     int get_x() { return (++(*x))--; } };
7. int *A::x(new int(20));
8. int main() {
9.     A *p=new A,b;
10.    cout<<b.get_x()<<" ";
11.    cout<<p->get_x();
12.    return 0; }
```

Programul compilează? DA ☒ NU ☐

Dacă DA ce se afișează pe ecran:

21 21

Dacă NU: de ce nu?

modificarea care îl face să meargă (o singură linie modificată, precizat nr linie modificat)

Barem: 0.5p pentru răspuns corect

0.3p pentru 21 20 sau 20 21

0.1p pentru 20 20

- V. Descrieți pe scurt mecanismul de tratare a excepțiilor (sintaxă, proprietăți, particularități, exemplu).

Barem: 0.1 pentru sintaxă +

0,1 cum se alege primul bloc catch corespunzător (după tipul expresiei) +

0,1 nu de face conversie de tip

0.1 explicare mecanism funcționare (nu sunt apeluri de funcții)

0.1 fără prostii și altele

Cum ar fi

0.1 pentru catch(...) +

0.1 pentru aruncarea excepțiilor din funcții +

- VI. Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ propuneți o (singură) modificare prin care programul devine corect.

```
1. #include <iostream>
2. using namespace std;
3. class B {
4.     int i;
5.     public: B() { i=50; }
6.     virtual int get_i() { return i; } };
7. class D: public B {
8.     int j;
9.     public: D() { j=47; }
10.    int get_j() {return j; } };
11. int main() {
12.    D *p=new B;
13.    cout<<p->get_i();
14.    cout<<((D*)p)->get_j();
15.    return 0; }
```

Programul compilează? DA ☐ NU ☒

Dacă DA ce se afișează pe ecran:

p e de tip *D nu poate pointa către *B

modificarea care îl face să meargă (o singură linie modificată, precizat nr linie modificat)

linia 12 D *p=new D;

Barem: 0.3 pentru eroarea identificată corect

0.2 pentru modificarea respectivă sau alta

VII. Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ propuneți o (singură) modificare prin care programul devine corect.

```

1.  #include <iostream>
2.  using namespace std;
3.  class A {
4.      int x_;
5.      public: int x() const { return x_; }
6.      void set_x(int x) { x_ = x; } };
7.  class B: public A { };
8.  class X {
9.      public: virtual void Print(A *a) {
10.         cout << "A.x = " << a->x() << endl; } };
11. class Y: public X {
12.     public: void Print(B *b) {
13.         cout << "B.x = " << b->x() << endl; } };
14. int main() {
15.     A* aptr = new B;
16.     aptr->set_x(3);
17.     Y* yptr = new Y;
18.     yptr->Print(aptr);
19.     return 0; }

```

Programul compilează? DA ☐ NU ☒ X
Dacă DA ce se afișează pe ecran:

Pe linia 18. Print() cere un parametru de tip B*, dar aptr este de tip A*, conversia la B* nefiind posibilă

modificarea care îl face să meargă (o singură linie modificată, precizat nr linie modificat)
În clasa Y, modificat parametrul lui Print() din B* în A*

Barem: 0.3 pentru eroare +
0.2 pentru modificare de mai sus sau
0.1 pentru orice altă modificare

VIII. Descrieți pe scurt caracteristicile fiecăreia și diferențele dintre o metodă statică și una nestatică.

Barem: 0.1 pentru diferența de sintaxă +

0.1 pentru obiectul implicit la metodele nestatice și lipsa lui la cele statice +

0.1 pentru accesul la elementele statice, respectiv nestatice din clasă +

0.1 pentru că metodele statice pot fi apelate în lipsa oricărui obiect din clasă, cele nestatice nu +

0.1 fără prostii

IX. Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ propuneți o (singură) modificare prin care programul devine corect.

```

1.  #include <iostream>
2.  using namespace std;
3.  class Base {
4.      protected: int x;
5.      public: Base(int i = 10) : x(i) {}
6.      int get_x() {
7.          cout << "Base ";
8.          return x; } };
9.  class Derived : public Base {
10.     public: Derived(int j) { x = j; }
11.     int get_x() {
12.         cout << "Derived ";
13.         return x; } };
14. int main() {
15.     Base *p1, *p2, o=50;
16.     p1 = &o;
17.     p2 = new Derived(11);
18.     cout << p1->get_x() + p2->get_x();
19.     return 0; }

```

Programul compilează? DA ☒ X NU ☐
Dacă DA ce se afișează pe ecran:

Base Base 61

Dacă NU: de ce nu?

modificarea care îl face să meargă (o singură linie modificată, precizat nr linie modificată și modificarea)

Barem: 0.5p pentru răspuns corect
0.2p pentru <2 clase> 61 sau
Base Base <valori apropiate>
0.1p - pentru <2 clase> <valori apropiate>

- X. Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ propuneți o (singură) modificare prin care programul devine corect.

```
1. #include<iostream>
2. using namespace std;
3. template<class X>
4. int functie(X *x, int y) {
5.     return x+y; }
6. int functie(int *x, int y) {
7.     return x-y; }
8. int main() {
9.     int *a= new int(45), b=27;
10.    cout<<functie(a,b);
11.    return 0; }
```

Programul compilează? DA ☐ NU ☒ X
Dacă DA ce se afișează pe ecran:

Dacă NU: de ce nu?

Dacă NU: de ce nu?

Funcția funcție() are tipul int, dar întoarce int* care nu se poate converti la int

modificarea care îl face să meargă (o singură linie modificată, precizat nr linie modificat)

Pe linia 6. schimbat tipul funcției în int*

Barem: 0.3 pentru eroare +

0.2 pentru modificare

- XI. Descrieți pe scurt caracteristicile fiecăreia și diferențele dintre compunere și moștenire.

dacă merge pe template și zice că e eroare 0.2

Barem: 0.1 definiție compunere +

0.1 definiție moștenire +

0.1 pentru crearea subcomponentelor, respectiv bazelor prin intermediul listei de inițializare a constructorului +

0.1 pentru diferența dintre drepturile de acces la elementele subcomponentelor (doar publice), respectiv la elementele bazei (publice și protected) +

0.1 pentru conversia de la derivată la bază (implicită), respectiv de la supraclasă la componentă (scrisă explicit prin supraincercare operatorului cast)

- XII. Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ propuneți o (singură) modificare prin care programul devine corect.

```
1. #include <iostream>
2. using namespace std;
3. struct X {
4.     int i;
5.     public: X(int ii = 5)
6.         { i = ii; cout<< i<< " ";}
7.     const int tipareste(int j)
8.         { cout<<i<< " "; return i+j; } };
9. int main() {
10.    X O (7);
11.    O.tipareste(5);
12.    X &O2=O;
13.    O2.tipareste(6);
14.    X* p=&O;
15.    cout<<p->tipareste(7);
16.    return 0; }
```

Programul compilează? DA ☒ X NU ☐
Dacă DA ce se afișează pe ecran:

7 7 7 7 14

Dacă NU: de ce nu?

modificarea care îl face să meargă (o singură linie modificată, precizat nr linie modificat)

Barem: 0.5p pentru răspuns corect

0.3p pentru cel puțin 3 valori

corecte din răspunsul corect

0.1p - cel puțin 1 valoare corectă din răspunsul corect

XIII. Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ propuneți o (singură) modificare prin care programul devine corect.

```

1. #include<iostream>
2. #include<vector>
3. class SimpleClass{
4.     int m_aMember;
5. public: SimpleClass(int i = 0)
6.     { std::cout << "C"<< std::endl;
7.       m_aMember = i; }
8.     SimpleClass(const SimpleClass &t)
9.     { std::cout << "CC" << std::endl; }
10.    ~SimpleClass() {
11.        std::cout << "D" << std::endl; };
12. int main(){
13.     std::vector<SimpleClass> vec;
14.     for(int i = 0; i < 2; i++) vec.push_back(i);
15.     return 0; }

```

Programul compilează? DA ☒ NU ☐

Dacă DA ce se afișează pe ecran:

C
CC
D
C
CC
CC
D
D
D
D

Barem:

Pentru output corect	0.5
Pentru C C D D	0.2
Pentru orice combinație de C-uri fără D-uri	0.1
Pentru C CC CC CC	0.3
Pentru C CC D C CC D	0.3

XIV. Scrieți o clasă care nu poate avea la un moment dat mai mult de 3 obiecte instanțiate.

Barem: 0.1 pentru clasă fără niciun obiect (clasă abstractă)

0.3 pentru clasă cu cel mult un obiect (singleton), câte 0.1 pentru constructor privat, funcție statică și alocare dinamică

0.5 pentru clasă cu cel mult două obiecte, câte 0.1 pentru constructor privat, funcție statică și alocare dinamică + 0.2 pentru restul

XV. Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ propuneți o (singură) modificare prin care programul devine corect.

```

1. #include <iostream>
2. using namespace std;
3. class cls {
4.     int x;
5. public: cls(int i=2) { x=i; }
6.     int set_x(int i) { int y=x;
7.                   x=i; return y; }
8.     int get_x(){ return x; } };
9. int main() {
10.    cls *p=new cls[15];
11.    for(int i=2;i<8;i++) p[i].set_x(i);
12.    for(int i=1;i<6;i++) cout<<p[i].get_x();
13.    return 0; }

```

Programul compilează? DA ☒ NU ☐

Dacă DA ce se afișează pe ecran:

22345

Dacă NU: de ce nu?

modificarea care îl face să meargă (o singură linie modificată, precizat nr linie modificat)

Barem: 0.5p pentru răspuns corect

0.2p pentru 12345

0.1p pentru oricare 5 valori afișate

XVI. Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ propuneți o (singură) modificare prin care programul devine corect.

```
1. #include<iostream>
2. using namespace std;
3. class A {
4.     int i;
5.     public: A(int x):i(x) { }
6.     int get_i() { return i; } };
7. class B {
8.     A a;
9.     int j;
10.    public: B(int x, A y) { j=x;
11.                           a=y; }
12.    int get_j() {return j; } };
13. int main() {
14.     A o1(5);
15.     B o2(6,o1);
16.     cout<<o1.get_i();
17.     cout<<o2.get_j();
18.     return 0; }
```

Programul compilează? DA ☐ NU ☒ X
Dacă DA ce se afișează pe ecran:

Constructorul lui B apelează implicit un constructor fără parametri al lui A (pentru creare subcomponentei a, care nu există)

modificarea care îl face să meargă (o singură linie modificată, precizat nr linie modificat)

Barem: 0.3 pentru eroare +
0.2 pentru apelul a(y) în lista de inițializare
sau
0.1 pentru orice altă modificare

XVII. Descrieți pe scurt clasele de tip template/șablon (sintaxă, particularități, utilitate, exemplu).

Sintaxa pe clase 0.1

Ca toate metodele sunt template 0.1

Ca pot fi și tipuri definite default (dacă nu sunt precizate se iau cele predefinite) 0.1

Crearea de clase parametrizate/refolosire de cod 0.1

Fără prostii 0.1

Altele: exemplu ar trebui să fie clar, nu în general; diferit de sintaxa

XVIII. Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ propuneți o (singură) modificare prin care programul devine corect.

```
1. #include <iostream>
2. using namespace std;
3. class A {
4.     protected: static int x;
5.     private: const int y;
6.     public: A(int i) { x=i; y=-i+3; }
7.     int put_x(A a) { return x+a.y; } };
8. int A::x=5;
9. int main() {
10.     A a(10);
11.     cout<<a.put_x(70);
12.     return 0; }
```

Programul compilează? DA ☐ NU ☒ X
Dacă DA ce se afișează pe ecran:
Câmpul y constant nu poate fi modificat prin atribuire

modificarea care îl face să meargă (o singură linie modificată, precizat nr linie modificat)

Pe linia 6., y inițializat prin lista constructorului

Barem: 0.3 pentru eroare +
0.2 pentru modificare de mai sus
sau
0.1 pentru orice altă modificare