Nume .....
Grupa .....

Nr. 1

23 iunie 2025
Programare orientată pe obiecte

Examen scris

Detalii: 1 punct din oficiu, 18 probleme fiecare valorează 0.5 puncte, timp de lucru 2 ore.

I. Spuneţi dacă programul de mai jos este corect. În caz afirmativ, spuneţi ce afişează, în caz negativ propuneţi 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 ☐
Dacă DA ce se afişează pe ecran:

Dacă NU: de ce nu?

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

0,3p Nu functioneaza – motivatia supraincarcarea f() in Derivata ascunde toate versiunile functiei din Baza, deci nu se poate apela f(string)
0,2p Modificare: exemplu, f()

II. Descrieţi folosirea pointerilor folosiţi împreună cu cuvântul cheie const (sintaxă, proprietăţi, particularităţi, exemplu).

0,1p x 5
a. sintaxa const int *p egal cu int const * p si diferit de int * const p
b. diferentele intre pointer const si
c. pointer catre chestie const
d. felul de atribuire: in stanga e cel mai "larg", nu se poate pointer const=pointer neconst
e. fara prostii; si altele: const int * const p

III. Spuneţi dacă programul de mai jos este corect. În caz afirmativ, spuneţi ce afişează, în caz negativ propuneţi o (singură) modificare prin care programul devine corect.

```
1. #include <iostream>
2. using namespace std;
3. class Cls
4. { int a;
5. public:
6. Cls(int x):a(x){cout<<a<<" ";}
7. ~Cls(){cout<<a<<" ";}
8. } A(10);
9.  void adauga() { static Cls B(20);}
10.      static Cls F(70);
11.      Cls C(30);
12.  int main(){Cls D(40);
13.  adauga();
14.  Cls E(50);
15.  static Cls G(80);
16.   cout<<"* "; return 0;}
```

Programul compilează? DA ☐ NU ☐
Dacă DA ce se afişează pe ecran:

Dacă NU: de ce nu?

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

0.5p - 10 70 30 40 20 50 80 * 50 40 80 20 30 70 10
0.2p – o combinatie aproape perfecta (de exemplu, se uita un static)
0.2p – 10 40 20 70 30 50 80 * 80 50 30 70 20 40 10 (standard fara static)
0.1p pentru macatr 4 valori corecte fara *

IV. Spuneţi dacă programul de mai jos este corect. În caz afirmativ, spuneţi ce afişează, în caz negativ propuneţi o (singură) modificare prin care programul devine corect.

```
#include <iostream>
using namespace std;
class Baza{ public: Baza(){cout<<"CB\n";} };
class Derivata1 : public Baza {
    public: Derivata1(){cout<<"CD1\n";}
            ~Derivata1(){cout<<"DD1\n";}   };
class Derivata2 : public Baza {
    public:     Derivata2(){cout<<"CD2\n";}
        virtual ~Derivata2(){cout<<"DD2\n";} };
class Derivata3 : virtual public Baza{
public: Derivata3(){cout<<"CD3\n";}  };
class Derivata4 : public Baza {public:Derivata4(){cout<<"DD4\n";} };
class Derivata5 : public Derivata1, Derivata2, protected Derivata3,
   public Derivata4 { public: Derivata5(){cout<<"Derivata5\n";} };
int main(){Derivata5 ob; }
```

Programul compilează? DA ☐ NU ☐
Dacă DA ce se afişează pe ecran:

Dacă NU: de ce nu?

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

0.5p – CB, CB, CD1, CB, CD2, CD3, CB, DD4, Derivata5, DD2, DD1
0.2p - CB, CD1, CB, CD2, CB, CD3, CB, DD4, Derivata5, DD2, DD1
0.1p – o varianta apropiata

V. Descrieţi particularităţile unui constructor definit cu atributul protected sau private (sintaxă, proprietăţi, particularităţi, exemplu).

0.1p x 5
a. sintaxa care sa compileze
b. sa mentioneze ca nu se pot instantia obiecte direct
c. sa dea exemple de instantieri (din clase derivate sau din functii statice)
d. mentionare singleton
e. fara prostii; din derivata se poate instantia

VI. Spuneţi dacă programul de mai jos este corect. În caz afirmativ, spuneţi ce afişează, în caz negativ propuneţi o (singură) modificare prin care programul devine corect.

```
1.  #include <iostream>
2.  using namespace std;
3.  class I { public:I(){cout<<"CI\n";}
4.           I(int x){cout<<x<<"ci\n";}
5.           void afis(){cout<<"i\n";} };
6.  class uni : virtual public I{
7.  public: uni(){cout<<"CU\n";}
8.   uni(int x):I(x){cout<<x<<" cu\n";}
9.  void afis(){cout<<"u \n";} };
10. class oras : virtual public I{
11. public: oras(){cout<<"CO\n";}
12.         oras(int x):I(x){cout<<x<<" co\n";}
13. void afis(){cout<<"Bucuresti";} };
14. class unibuc : public uni,public oras{ public: unibuc(){cout<<"CUB\n";}
15. unibuc(int x):I(x){} };
16. int main(){
17. unibuc ob;
18. ob.afis();
19. unibuc ob2(10);}
```

Programul compilează? DA ☐ NU ☐
Dacă DA ce se afişează pe ecran:

Dacă NU: de ce nu?

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

0.3p - ob.afis() e ambiguu
0.3p – modificare: ob.uni::afis() (sau din celelalte clase); sau 0.2p daca adauga functia afis() in derivata

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 Cls{ public: void afis(){cout<<"1 ";}
4.  Cls operator+(Cls ob){cout<<"2 "; return Cls();}
5.  Cls operator-(Cls ob){cout<<"3 "; return Cls();}};
6.  class Cls2 : public Cls{
7.  public: void afis(){cout<<"4 ";}
8.  Cls2 operator+(Cls2 ob){cout<<"5 "; return Cls2();} };
9.  int main() {
10. Cls a;
11. Cls2 d,e,f;
12. (d + e).afis();
13. (d - e).afis();
14. (a + d).afis();
15. (d + a).afis();
16. }
```

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

Dacă NU: de ce nu?

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

0.3p – nu exista operator+ (Cls2, Cls)
0.2p – d+e sau alta modificare in randul 15 sau 0.2p – adaugarea unui op+

VIII. Descrieți particularitățile metodelor statice considerând în special folosirea lor la moștenirea multiplă. (sintaxă, proprietăți, particularități, apelare).

0,1p x 5

a. sintaxa corecta
b. apel corect
c. ca nu se poate face virtualizare pe ele (nu sunt in vtable)
d. ca se redefinesc in derivata cu aceeasi semnatura
e. fara prostii (nu sunt diferente intre mostenire si MM)

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.  int main(){
4.  const char* sFirst = "Examen\0 2025\n";
5.  char sSecond[64];
6.  const char* sSrc = sFirst;
7.  char* sDst = sSecond;
8.  while (*sDst++ = *sSrc++);
9.  std::cout << sSecond;
10. }
```

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

Dacă NU: de ce nu?

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

0.5p – afisare „Examen"
0.1p – afisare „Examen 2025" sau „2025"

X. Spuneţi dacă programul de mai jos este corect. În caz afirmativ, spuneţi ce afişează, în caz negativ propuneţi o (singură) modificare prin care programul devine corect.

```
1. #include<iostream>
2. using namespace std;
3. int main()
4. {int a = 1;
5. int b = ++a;
6. const int *a_ptr = &a;
7. int *const b_ptr = &b;
8. *b_ptr += 4;
9. *a_ptr += 5;
10.      std::cout << *b_ptr << std::endl;
11.      std::cout << *a_ptr << std::endl;
12.return 0; }
```

Programul compilează? DA ☐ NU ☐
Dacă DA ce se afişează pe ecran:

Dacă NU: de ce nu?

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

0.3p - Codul Nu Compileaza

Explicatii: a_ptr pointeaza la un int const, nu se poate schimba continutul din pointer.

0.2p - Daca se vrea schimbarea valorii atunci trebuie eliminat "const"

XI. Descrieţi particularităţile operatorului typeid. (sintaxă, proprietăţi, particularităţi, motivaţie).

0.1p x 5

a. sintaxa/ exemplu corect cu mostenire, virtualizare etc
b. ca se foloseste la RTTI deci virtualizare, mostenire, pointer sau referinta
c. ca e un text care se produce in functie de compilator, prefixul e pentru clase de baza si sufixul pt derivate
d. similaritati cu dynamic_cast
e. fara prostii

XII. Spuneţi dacă programul de mai jos este corect. În caz afirmativ, spuneţi ce afişează, în caz negativ propuneţi o (singură) modificare prin care programul devine corect.

```
1.  #include <iostream>
2.  using namespace std;
3.  class Baza{ public: virtual ~Baza(){}};
4.  class D1 : virtual public Baza{};
5.  class D2 : public D1{};
6.  class D3 : virtual public Baza{};
7.  class D4 : public D1, public D3{};
8.  int main() {
9.  D4 ob;
10. Baza& re = ob;
11. try{ throw re; }
12. catch(D1& o){cout<<"D1\n";}
13. catch(D2& o){cout<<"D2\n";}
14. catch(D3& o){cout<<"D3\n";}
15. catch(D4& o){cout<<"D4\n";}
16. catch(Baza& o){cout<<"Baza\n";}
17. }
```

Programul compilează? DA ☐ NU ☐
Dacă DA ce se afişează pe ecran:

Dacă NU: de ce nu?

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

0.5p – afiseaza Baza
0.2p – afiseaza D4
0.1P – afiseaza D1, sau D2, sau D3

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.  using namespace std;
3.  class B { int b;
4.  public: B(int p = 1) { b = p; } };
5.  class D : public B { int* d;
6.  public:     D(int p) { d = new int; *d = p; }
7.  D(const D& s) : B(s) { d = new int; *d = *(s.d); }
8.      ~D() { delete d; }
9.      void set(int p) { *d = p; }};
10.
11. int main() {
12.     D o1(2), o2(3);
13.     o1 = o2;
14.     o2.set(4);
15.     return 0;}
```

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

Dacă NU: de ce nu?

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

0.5 compileaza si se intoarce cod diferit de 0 catre OS
0.2 compileaza si nu afiseaza nimic sau compileaza si intoarce 0 la OS
0.3p - * Compilează? Codul compilează, dar la rulare va apărea o eroare de dublă eliberare a memoriei, de aceea return code-ul nu va fi 0.
* Explicație: Operatorul de atribuire nu este suprascris, deci este folosit cel implicit, care face copiere bit cu bit. Astfel, atât o1, cât și o2 vor avea pointerul d către aceeași zonă de memorie, iar la distrugere se va încerca eliberarea de două ori a aceleiași zone.
0.2p - * Rezolvare: Se suprascrie operatorul de atribuire pentru a face deep-copy pe date, nu doar copiere de pointer sau 0.2p = se comenteaza linia 13 o1 = o2;

XIV. Descrieți noțiunea de destructor virtual pur în C++. (sintaxă, proprietăți, particularități, motivație).

0,1p x 5
a)sintaxa pt destructor virtual pur,
b) ca se da implementarea chiar daca e virtual pur (diferenta fata de alte functii virtual pure)
c) motivatie/de ce: ca sa nu se poata instantia interfata, definitia interfetei
d) motivatie/de ce: previne object slicing
e) fara prostii

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 B {public:
4.  virtual B* fv() { return this; }
5.  int adun(int p) { return p + 1; }};
6.  class D : public B {public:
7.  virtual D* fv() { return this; }
8.  int adun(int p) { return p + 2; }};
9.  int main() {
10. B* p = new D;
11. int x = p->fv()->adun(1);
12. std::cout << x << "\n";
13. return 0;}
```

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

Dacă NU: de ce nu?

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

0.5p - Afișează: 2
0.2p – Afiseaza: 3
* Explicație: Deși fv() este virtuală și returnează un pointer la obiectul de tip D, metoda adun() nu este virtuală, deci se apelează versiunea din clasa de bază (B), nu cea suprascrisă în D

XVI. Spuneţi dacă programul de mai jos este corect. În caz afirmativ, spuneţi ce afişează, în caz negativ propuneţi o (singură) modificare prin care programul devine corect.

```
1.  #include<iostream>
2.  using namespace std;
3.  class C {int c;
4.  public: C(int p = 1) { c = p; }
5.          int& get() const { return c; } };
6.
7.  int f(C op) { return op.get(); }
8.
9.  int main() {
10. C o1;
11. int x = f(o1);
12. std::cout << x << "\n";
13. return 0; }
```

Programul compilează? DA ☐ NU ☐
Dacă DA ce se afişează pe ecran:

Dacă NU: de ce nu?

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

0.3p - Compilează? NU.
* Explicaţie: Nu compilează, deoarece funcţia get() este declarată const dar returnează o referinţă non-const la membru, ceea ce ar permite modificarea unui membru dintr-un obiect constant.
0.2p - * Rezolvare: Se adaugă const la tipul returnat.sau altele

XVII. Definirea copy-constructorului de către programator. (sintaxă, proprietăţi, particularităţi, motivaţie).

0.1p x 5
a. cand avem alocare dinamica de memorie in constructor/dezalocare in destructor
b. cand avem const ca data membra verif
c. sintaxa
d. daca se defineste dispare constructorul fara parametri, ca ar trebui redefinit si operatorul =, ca nu se apeleaza daca se face apel prin referinta, etc
e. fara prostii

XVIII. Spuneţi dacă programul de mai jos este corect. În caz afirmativ, spuneţi ce afişează, în caz negativ propuneţi o (singură) modificare prin care programul devine corect.

```
#include <iostream>
using namespace std;
class C { int a;
          static int x;
public:
     C(int a = 22) { x++; this->a=a; }
     static int f() {return x;}
     int getA(){return a;}
}a;
int C::x=20;
int main() {
   C a(33),b;
   cout<<"instantieri C: "<<C::f()<<" val elem:"<<::a.getA();
   return 0; }
```

Programul compilează? DA ☐ NU ☐
Dacă DA ce se afişează pe ecran:

Dacă NU: de ce nu?

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

0.5p – afiseaza „instantieri C: 23 val elem:22"
0.1p – afiseaza „instantieri C: 22 val elem:21" sau 33