1. (0.5p) Care sunt valorile afișate în urma execuției codului de mai jos:

```
class Student{
  int id;
public:
  Student(){
    cout << "constr Student()";
    this->id = 0;
  }
  Student(int i){
    cout << "constr Student(int i)";
    this->id = i;
  }
  ~Student(){}
};
```

```
void main(){
    Student s1;
    Student s2 = Student();
    Student s3 = Student(3);
}
```

a) constr Student()
   constr Student(int i)
b) constr Student()
   constr Student()
   constr Student(int i)
c) constr Student()
   constr Student()
d) constr Student(int i)
   constr Student(int i)

2. (0.5p) Care dintre următoarele afirmații sunt false:
   a) O metodă privată (*private*) poate fi accesată doar în interiorul clasei care o definește
   b) O metodă protejată (*protected*) poate fi accesată doar în interiorul clasei care o definește
   c) O metodă publică (*public*) poate fi accesată doar în interiorul clasei care o definește
   d) O metodă protejată (*protected*) poate fi accesată în interiorul clasei care o definește

3. (1p) Completați afirmațiile de mai jos:
   a) Metodele accesor de tip *get* ale unei clase returnează valori de tipul: .....
   b) Constructorul cu parametrii al unei clase derivate efectuează doi pași: ..... și .....

4. (1p) Fie clasa ***Carte*** cu atributul *titlu* de tip string. Definiți constructorul cu parametrii și cel de copiere ai clasei ***Manual***, derivată din clasa ***Carte***, știind că un manual se caracterizează prin *titlu* (de tip string) și *nrLecții* (de tip număr întreg).

5. (1p) Identificați și explicați 3 erori de sintaxă în următoarea secvență de cod:

```
class A{
  private:
    int nr;
    int getNr() { return this->nr; }
  public:
    A(int x) { this->x = nr; }
};
A a = A(3);
A b = a.getNr();
```

6. (1p) Se consideră clasele *Profesor*, *Student*, *SalăDeCurs*, *Persoană* și *Disciplină*. Judecând după denumiri, identificați și argumentați o relație de moștenire și una de agregare între aceste clase; reprezentați aceste relații prin diagramă UML.

7. (1p) Fie următoarea secvență de cod:

```
class A{
        int x;
public:
        A(int val){ this->x = val;}
};

class B : public A{
        int y;
public:
        B(int v1, int v2) : A(v1){
                this->y = v2;
        }
};
```

Alegeți variantele în care se realizează *object slicing*:

a) `A a1 = A(1); B b1 = B(4, 5); a1 = b1;`
b) `A a2 = A(2); B b2 = B(6, 7); b2 = a2;`
c) `A* a3; B* b3 = new B(8, 9); a3 = b3;`
d) `A* a4 = new A(9, 10); B* b3 = a4;`

8. (0.5p) Implementați următoarea cerință: creați un vector din biblioteca stl care să conțină 3 cărți: (Beloved, 2015), (Jazz, 2016), (Quarantine, 2014), presupunând că există definită clasa ***Carte*** cu atributele *titlu* (de tip string) și *anApariție* (De tip număr întreg).

9. (0.5p) Fie o clasă ***NumărComplex*** cu metoda *add(NumărComplex b)* – care adună numărul complex curent cu numărul complex *b* și returnează numărul complex sumă rezultat - și metoda *mul(NumărComplex b)* – care are un efect similar, dar pentru operația de înmulțire. Fie expresia:

```
NumărComplex s; NumarComplex t = ((s.add(b)).add(c)).mul(d);
```

Definiți metoda *add* știind că un număr complex are două atribute: *real* și *imaginar* de tip numere reale.

10. (3p) Presupunem că a fost dezvoltată clasa ***NumărComplex*** cu atributele *real* și *imaginar* de tip numere reale. Dezvoltați o aplicație (folosind principiile POO) pentru ora de matematică care să permită:
    a. Crearea și afișarea unei mulțimi de numere complexe
    b. Determinarea și afișarea intersecției a două mulțimi de numere complexe