Barem Teorie

T1) Am definita clasa Irat de numere irationale (cu parte reala si parte imaginara). Instantiez obiectele x, y, z de acest fel. Ce trebuie facut ca bucata de cod urmatoare sa compileze:
y=2*x;
x=x+1;
z=y/2;
descrierea conceptului/conceptelor, sintaxa, proprietati, restrictii.

5 variante dar toate asemanatoare
0.1 p pentru mentionat redefinirea operatorilor
0.2 p pentru redefinirea corecta a operatorului * (se poate face doar ca functie independenta)
0.1 p pentru redefinirea celorlalti operatori (+, /)
0.1 p pentru alte proprietati ale redefinirii operatorilor
(trebuie neaparat sintaxa corecta macar pentru unul din operatorii *, +, /) altfel max 0.3
sintaxa nu e data max 0.3, sintaxa gresita (Irat operator/(Irat & x)) max 0.2

T2) Dati cat mai multe variante de a se modifica starea (variabilele de instanta) unui obiect constant care apeleaza o functie constanta.
0.2 p pentru fiecare din urmatoarele (const_cast, cast away constness, mutable, volatile) cu un maxim de 0.5
prostii -0.1p pentru fiecare prostie
daca se precizeaza ca din functii const NU se poate modifica starea obiectului max 0.1p

T3) Cum functioneaza operatorul de atribuire implicit (dat de compilator), al unei clase compuse (cu date dintr-o clasa de baza),si cum trebuie scris de programator, operatorul de atribuire pentru aceeasi clasa compusa? Sintaxa.

0.1 op= implicit -copiere bit cu bit
0.1 op=implicit -apel op= pt date membre (sau baza!! -pt cei care au inteles derivare)
0.1. suprascriere ca metoda
0.1 suprascriere -cand exista date alocate dinamic
0.1 op = suprascris nu apeleaza implicit op= pt datele membre -> trebuie apelat explicit
0.1 sintaxa op = antet(C& operator=(C&)){return *this }
/alep op= membre(data=param.data)
/apel op=baza(baza::operator=(param)) (pt cei care au inteles gresit derivare)
ultimele doua sunt sau exclusiv

T4) Descrieti specializarea explicita pentru sabloane (template-uri) de clase.
Sintaxa, proprietati, observatii.
0.1p specifice pentru anumite tipuri de date
0.1p sintaxa: template<>class Nume<Tip_particular>
0.1p exemplu
0.1p nu se pastreaza nimic din clasa template pe care o specializeaza, deci trebuie rescrise metodele
0.1p e prioritara fata de template pt instantiere cu acelasi tip de data 0.1p
0.1p fara prostii

T5) Descrieti cum se poate re-arunca o exceptie: sintaxa, proprietati, restrictii, utilizare.
0.1p **throw;**
0.2p ca e try in try si primul throw x iar in primul catch se face throw; si se prinde in al doilea catch
0.1p ca se face cand se vrea re-procesarea unei exceptii din alt punct de vedere
0.1p altele: ca se poate face din functii sau nu, etc.
0.1p fara prostii

T6) Sa se scrie cod pentru urmatoarea situatie:

sa se construiasca o clasa CLS care sa aibe ca date de instanta cel putin doi intregi unul constant c si unul normal i.

se citeste n de la tastatura (va fi intotdeuna mai mare decat 10)

sa se instantieze n obiecte din aceasta clasa CLS cu restrictiile urmatoare:

1. primele 3 obiecte vor avea initializate c si i cu c=2 si i=3

2. obiectele de la 5 la 6 vor avea c si i la valoarea 1

3 obiectele de la 7 la n vor avea c=numarul obiectului respectiv, i va fi n-c, adica al optulea obiect va avea c=8, al noualea obiect va avea c=9, etc.

Sa se descrie notiunea de POO care va ajuta sa rezolvati restrictiile 1,2,3

5 variante similar cu unele mici diferente

- incalcarea conceptelor de POO (e.g. atribuirea de valori catre variabile const dupa initializare sau incalcarea encapusularii informatiei - acesarea in main a unei proprietati private) - 0p
- Mentionarea si utilizarea corecta in cod a listei de initializare a constructorului pentru initializarea variabile const 0.2p:
  - 0.1p mentionare
  - 0.1p utilizare
- Implementarea instantierii celor n obiecte 0.3p:
  - 0.2p instantierea obiectelor conform restrictiilor (se acorda punctaj partial)
  - 0.1p salvarea obiectelor intr-o structura de tip colectie:
    - daca se folosesc pointeri cu new/delete (i.e. alocarea dinamica): 0.05p pentru alocarea propriu-zisa si 0.05 pentru dezalocare
    - daca se foloseste vector => 0.1p

Exercitiul 1

0,4 nu compileaza cu explicatii
0,1 modificarea

var 1

```
class C{
int c;
public:  C(int p=1){c=p;}
int & get()const{return c;}
};
int f(C op){return op.get();}
int main(){
C o1;
int x=f(o1);
return 0;
}
```

Explicatii : nu pot intoarce const int & in int &;
Modificare: adaug const la referinta intoarsa

---

var 2

```
class C{
int c;
public:  C(int p=1){c=p;}
const int & get(){return c;}
};
int f(const C*op){return op->get();}
int main(){
C o1;
int x=f(&o1);
return 0;
}
```

Explicatii: p protejeaza zona, nu poate apela metoda neconstanta;
Modificare: adaug const la metoda

---

var 3

```
class C{
int c;
public:  C(int p=1){c=p;}
const int & get(){return c;}
};
int f(const C & op){return op.get();}
int main(){
C o1;
```

```
int x=f(o1);
return 0;
}
```

Explicatii: r protejeaza zona, nu poate apela metoda neconstanta;
Modificare: adaug const la metoda

```
var 4
class C{
int c;
public:  C(int p=1){c=p;}
const int & get()const{return c++;}
};
int main(){
const C o1;
int x=o1.get();
return 0;
}
```

Explicatii: metoda constanta nu poate modifica date obiectului implicit;
Modificare: intoarce c

```
var 5
class C{
int c;
public:  C(int p=1){c=p;}
const int & get(){return c;}
};
int f(const C op){return op.get();}
int main(){
C o1;
int x=f(o1);
return 0;
}
```

Explicatii: obiect constant nu poate apela metoda neconstanta ;
Modificare: adauga const la metoda;

Exercitiul 2

0,2 alegere corecta fv()+ rezultat
0,2 alegere corecta adun(1)+rezultat
0,1 explicatii

var 1

```
class B{
public: virtual B * fv(){return this;}
        int adun(int p){return p+1;}
};
class D:public B{
public: virtual D * fv(){return this;}
        int adun (int p){return p+2;}
};
int main(){
B *p =new D;
int x=p->fv()->adun(1);
return 0;
}
```

Barem : fv intoarce corect pointerul din D dar metoda adun nu e virtuala !!!deci se evalueaza la compilare si se apeleaza cea din baza
Raspuns:x=2

---

var 2

```
class B{
public: virtual B*  fv(){return this;}
        virtual int adun(int p){return p+1;}
};
class D:public B{
public: virtual B*  fv(){return this;}
        virtual int adun (int p){return p+2;}
};
int main(){
B *p =new D;
int x=p->fv()->adun(1);
return 0;
}
```

Barem : fv intoarce pointer din B dar care contine adresa de ob din D!!!, care va alege varianta din D
Raspuns:x=3

---

var 3

```
class B{
```

```
public: virtual B*  fv(){return this;}
        int adun(int p){return p+1;}
};
class D:public B{
public: virtual B*  fv(){return this;}
       int adun (int p){return p+2;}
};
int main(){
B *p =new D;
int x=p->fv()->adun(1);
return 0;
}
```
Barem : fv intoarce pointer din B, care contine adresa de ob din D , dar adun nu e virtual -o alege pe cea din B
Raspuns:x=2

---

var 4
```
class B{
public: virtual B * fv(){return this;}
       virtual int adun(int p){return p+1;}
};
class D:public B{
public: virtual D * fv(){return this;}
       virtual int adun (int p){return p+2;}
};
int main(){
B *p =new D;
int x=p->fv()->adun(1);
return 0;
}
```
Barem : fv intoarce pointer din D si va alege adun din D
Raspuns:x=3

---

var 5.
```
class B{
public:  B * fv(){return this;}
        virtual int adun(int p){return p+1;}
};
class D:public B{
public:  B * fv(){return this;}
        virtual int adun (int p){return p+2;}
};
int main(){
B *p =new D;
int x=p->fv()->adun(1);
return 0;
```

}

Barem: aleg fv din B , intoarce pointer din B care contine adresa de ob din D, se alege adun din D
Raspuns:x=3

Exercitiul 3

var 1

```
class B{
        int b;
public: B(int p=1){b=p;}
};
class D: public B{
         int *d;
 public: D(int p){d=new int; *d=p;}
         D(const D& s):B(s){d=new int; *d=*(s.d);}
         ~D(){delete d;}
         void set(int p){*d=p;}
};

int main()
{D o1(2),o2(3);
 o1=o2;o2.set(4);
return 0;
}
```

Explicatii : lipseste op = deci se apeleaza operatorul = implicit care copiaza bit cu bit si o1 si o2 vor avea aceasi valoare pt d;
In zona comuna se va pune 4;
Eroare la eliberearea dubla a zonei

var 1
0,1 constructor de initializare baza
0,1 nu e supraincarcat op=/construct copiere
0,1 implicit ->copiere bit cu bit
0,1 zona comuna adresata de acelasi pointer
0,1 eroare la dubla eliberare a zonei comune

---

var 2

```
class B{
        int b;
public: B(int p=1){b=p;}
};
class D: public B{
         int *d;
 public: D(int p){d=new int; *d=p;}
         D(const D& s):B(s){d=new int; *d=*(s.d);}
         void set(int p){*d=p;}
};

int main()
```

```
{D o1(2),o2(o1);
 o1=o2;o2.set(4);
return 0;
}
```

Explicatii : lipseste op = deci se apeleaza operatorul = implicit care copiaza bit cu bit si o1 si o2 vor avea aceasi valoare pt d;
In zona comuna se va pune 3; lipseste si destructorul deci eroarea nu va fi semnalata la executie
var 2
0,1 constructor de initializare baza
0,1 nu e supraincarcat op=/construct copiere
0,1 implicit ->copiere bit cu bit
0,1 zona comuna adresata de acelasi pointer
0,1 nu da eroare daca nu se elibereaza zona comuna
---
var 3

```
class B{
	int b;
public: B(int p=1){b=p;}
};
class D: public B{
	int *d;
public: D(int p):B(p){d=new int; *d=p;}
	D(const D& s){d=new int; *d=*(s.d);}
	D & operator=(const D & s){d=new int; *d=*(s.d);return *this; }
	~D(){delete d;}
	void set(int p){*d=p;}
};

int main()
{D o1(2),o2(o1);
 o1=o2;
return 0;
}
```

Explicatii :!!! constructorul de copiere pt o2 -va apela constructorul de initializare din baza cu val implicita p=1 deci o2.b=1;
op= nu copiaza si data b deci o1.b=2;
var 3
0,1 constructor de initializare baza
0,2 constructorul de copiere explicit -nu apeleaza implicit cc de baza (o2.b=1)
0,2 op= implicit -nu apeleza implicit op= din baza (o1.b=2)
---

var 4

```
class B{
	int b;
public: B(int p=1){b=p;}
};

class D: public B {
	int *d;
public: D(int p):B(p){d=new int; *d=p;}
	D & operator=(const D & s){d=new int; *d=*(s.d);return *this; }
	~D(){delete d;}
	void set(int p){*d=p;}
};
int main()
{D o1(2),o2(o1);
 o2.set(3);
return 0;
}
```

Explicatii : Constructor de copiere implicit , copiere bit cu bit deci o1 si o2 vor avea aceeasi valoare pt d;
in zona comuna se va pune valoarea 3;
Eroare la eliberarea dubla a zonei
var 4
0,1 constructor de initializare baza
0,1 nu e supraincarcat op=/construct copiere
0,1 implicit ->copiere bit cu bit
0,1 zona comuna adresata de acelasi pointer
0,1 eroare la dubla eliberare a zonei comune

---

var 5

```
class B{
	int b;
public: B(int p=1){b=p;}
};

class D: public B {
	int *d;
public: D(int p):B(p){d=new int; *d=p;}
	D & operator=(const D & s){d=new int; *d=*(s.d);return *this; }
	void set(int p){*d=p;}
};
int main()
{D o1(2),o2(o1);
 o2.set(3);
return 0;
```

```
}
```

Explicatii :Constructor de copiere implicit , copiere bit cu bit deci o1 si o2 vor avea aceeasi valoare pt d si b=2;
in zona comuna pt d se va pune valoarea 3; lipseste insa destructorul si eroarea nu va fi detectata la executie
var 5
0,1 constructor de initializare baza
0,1 nu e supraincarcat op=/construct copiere
0,1 implicit ->copiere bit cu bit
0,1 zona comuna adresata de acelasi pointer
0,1 nu da eroare daca nu se elibereaza zona comuna
---
var 6

```
class B{
	int b;
public: B(int p=1){b=p;}
};

class D: public B {
	int *d;
public: D(int p):B(p){d=new int; *d=p;}
	D & operator=(const D & s){d=new int; *d=*(s.d);return *this; }
	~D(){delete d;}
	void set(int p){*d=p;}

};
int main()
{D o1(2),o2(o1);
 o1=o2;o2.set(3);
return 0;
}
```

Explicatii : Constructor de copiere implicit , copiere bit cu bit deci o1 si o2 vor avea aceeasi valoare pt d=2 si b=1;
!!! op= aloca zona noua pt o1.d ; doar o2.d va fi 3;
var 6
0,1 constructor de initializare baza
0,1 nu e supraincarcat op=/construct copiere
0,1 implicit ->copiere bit cu bit
0,1 zona comuna adresata de acelasi pointer
0,1 op= definit aloca zona noua -> nu e eroare

Exercitiul 4

var 1

```
#include <iostream>
using namespace std;

class C {
   float z;
public:
   C() { z = 1.3; }
   float op(float x, float y, float t) {
      return x + y + z + t;
   }
   float op(float x, float y = 1.0) {
      return x + y + z;
   }
   float op() {
      return z;
   }
};

int main() {
   C c;
 float i,j, k;
   i=c.op(1);
   j=i+c.op(2.2, 4.8);
   k=c.op(2.2, 3.5, 4);

   return 0;
}

//Compileaza
//Se afiseaza
//3.3
//11.6
//11

//compileaza si valori rezonabile 0.1
//pentru 3.3  0.2
//pentru 11.6 0.1
//pentru 11   0.1
```

---

var 2

```
#include <iostream>
using namespace std;

class C {
   float z;
public:
   C() { z = 1.3; }
   float op(float x, float y, float t) {
      return x + y + z + t;
   }
   float op(float x, float y = 1.0) {
      return x + y + z;
   }
   float op(float x) {
      return x + z;
   }
};

int main() {
   C c;

 float i,j, k;
   i=c.op(1.2);
   j=i+c.op(2.2, 4.8);
   k=c.op(2.2, 3.5, 4);

   return 0;
}
//Nu compileaza 0.1
//Eroarea apare deoarece apelul din main de la linia 22 (primul apel catre op din main) este ambiguu
//0.2
//Se considera candidati valizi atat metoda op de la linia 11 cat si cea de la linia 14
//Solutia pentru a rezolva eroarea este sa indepartam valoarea default de la linia 11 pentru variabila y (a doua metoda op din clasa C)
// 0.2
```

---

var 3

```
#include <iostream>
using namespace std;

class C {
```

```
    float z;
public:
    C() { z = 1.3; }
    float op(float x, float y, float t) {
        return x + y + z + t;
    }
    float op(float x = 1.0, float y) {
        return x + y + z;
    }
    float op() {
        return z;
    }
};

int main() {
    C c;

 float i,j, k;
    i=c.op(1.2);
    j=i+c.op(2.2, 4.8);
    k=c.op(2.2, 3.5, 4);

    return 0;
}
//Nu compileaza 0.1
//Parametrii default trebuie sa apara ultimii in enumeratia de la linia 11 (a doua metoda op din clasa C) 0.2
//Solutia pentru a rezolva eroarea este sa indepartam valoarea default de la linia 12 pentru variabila x si sa o punem la y, sau sa inversam x si y intre ele in definitia metodei
//0.2
```

---

var 4

```
#include <iostream>
using namespace std;

class C {
    float z;
public:
    C() { z = 1.3; }
    float op(float x = 2.0, float y = 1.2, float t = 1.5) {
        return x + y + z + t;
    }
    int op(int y) {
        return y + z;
```

```
    }
    double op(double y) {
        return y + z;
    }

};

int main() {
    C c;
 float i,j, k;
    i=c.op();
    j=i+c.op(1.2);
    k=c.op(2);

    return 0;
}
//Compileaza
//Se afiseaza
//6
//8.5
//3

//compileaza si valori rezonabile 0.1
//pentru 6  0.1
//pentru 8.5 0.2
//pentru 3   0.1
```

---

var 5

```
#include <iostream>
using namespace std;

class C {
    float z;
public:
    C() { z = 1.3; }
    float op(float x = 2.0, float y = 1.2, float t = 1.5) {
        return x + y + z + t;
    }
    float op(float y) {
        return y + z;
    }
    double op(double y) {
        return y + z;
    }
```

```
};

int main() {
    C c;

    cout << c.op() << "\n";
    cout << c.op(1.2) << "\n";
    cout << c.op(1) << "\n";

    return 0;
}
```
//Nu compileaza 0.1
//Problema apare la apelul functiilor, mai exact la linia 24, ultimul apel este ambiguu, posibili candidati sunt atat metoda op de la linia 12, cat si metoda op de la linia 15 (penultima si ultima metoda op din clasa C)
//0.2 p daca zice de linia 23 0.1p si atat

//O posibila solutie este sa modificam semnatura metodei de la linia 12 incat sa preia un parametru int y si nu float y
// 0.2p

Exercitiul 5

var 1

```
#include <iostream>
using namespace std;

class B {
protected:
   static int x;
   int offset;
public:
   B()
   {x++; offset = 100;}
   ~B() { x--; }
   static int get_x() { return x; }
   int get_offset() { return offset; }
   int f() { return (x + offset) / 2; }
};
int B::x = 0;

class D : public B {
public:
   D() { x++; }
   ~D() { x--; }
   int f() { return ( (x + offset) / 2 + 1); }
};

void func(B* q, int n) {cout << q->get_x() << " ";
   for(int i = 0; i < n; i++) cout << q[i].f() << " ";
   cout<<"\n";
}

int main()
{
   B* p = new B[2]; func(p, 2); delete[] p;
   p = new D; func(p, 1); delete p;
   cout << D::get_x();    return 0;
}
//Compileaza  0.1p
//Se afiseaza
//2 51 51    0.2
//2 51       0.1
//1          0.1
-----------------------------------------------------
```

var 2

```
#include <iostream>
using namespace std;

class B {protected:
   static int x;
   int offset;
public:
   B()    { x++;offset = 100; }
   ~B() { x--; }
   static int get_x() { return x; }
   int get_offset() { return offset; }
    int f() { return (x + offset) / 2; }
};
int B::x = 0;

class D : public B {public:
   D() { x++; }
   ~D() { x--; }
   int f() { return (x + offset) / 2 + 1; }
};

void func(B* q, int n){cout << q->get_x() << " ";
   for(int i = 0; i < n; i++) cout << q[i].f() << " ";
   cout<<"\n";}

int main()
{
   B* p = new B[2];func(p, 2);delete[] p;
   p = new D;func(p, 1); delete p;
   cout << D::get_x();
}
//Nu compileaza 0.1
//deoarece incercam sa apelam variabila offset in
//cadrul unei functii statice f() din clasa B  0.2
//Pentru a face programul sa compileze trebuie sa scoatem
//identificatorul static pentru aceasta functie 0.2
```

---

var 3

```
#include <iostream>
using namespace std;
```

```
class B {
protected:
   static int x;
   int offset;
public:
   B() {x++; offset = 100; }
   ~B() { x--; }
   virtual static int get_x() { return x; }
   int get_offset() { return offset; }
   virtual int f() { return (x + offset) / 2; }
};
int B::x = 0;

class D : public B {public:
   D() { x++; }
   ~D() { x--; }
   int f() { return (x + offset) / 2 + 1; }
};

void func(B* q, int n){cout << q->get_x() << " ";
   for(int i = 0; i < n; i++)cout << q[i].f() << " ";
   cout<<"\n";}

int main()
{
   B* p = new B[2];func(p, 2);delete[] p;
   p = new D;func(p, 1);delete p;
   cout << D::get_x();
}
//Nu compileaza  0.1
//functia get_x din B nu poate fi declarata si virtual si static (nu exista functii virtuale statice)
//0.2
//Pentru ca programul a compileze trebuie sa eliminam identificatorul virtual pentru get_x
//0.2
```

---

var 4

```
#include <iostream>
using namespace std;

class B {
protected:
   static int x;
   int offset;
```

```
public:
   B(){ x++;offset = 100;}
   ~B() { x--; }
   static int f() { return x; }
   int get_offset() { return offset; }
   virtual int f() { return (x + offset) % 2; }
};
int B::x = 0;

class D : public B {public:
   D() { x++; }
   ~D() { x--; }
   int f() { return (x + offset) % 2 + 1; }
};

void func(B* q, int n){
   for(int i = 0; i < n; i++) cout << q[i].f() << " ";
   cout<<"\n";
}

int main()
{
   B* p = new B[2];func(p, 2);delete[] p;
   p = new D;func(p, 1);delete p;
}
//Nu compileaza 0.1
//Nu putem avea in aceeasi clasa o functie cu aceeasi semnatura
//care sa fie si statica si non-statica in acelasi timp 0.2
//Pentru a compila trebuia de exemplu sa dam un argument x functiei f()
//de la linia 24 pentru a le diferentia semnatura (x va fi aici o variabila locala considerata in loc
de x-ul static)
//Sau comentam linia ce contine varianta de functie statica
//Sau comentam linia ce contine varianta virtual int f()  0.2
```

---

var 5

```
#include <iostream>
using namespace std;

class B {
protected:
   static int x;
   int offset;

public:
```

```
   B()    { x++; offset = 100; }
   ~B() { x--; }
   static int get_x() { return x; }
   int get_offset() { return offset; }
   static int f() { return (x + get_offset()) % 2; }
};
int B::x = 0;

class D : public B {public:
   D() { x++; }
   ~D() { x--; }
   int f() { return (x + offset) % 2 + 1; }
};

void func(B* q, int n){
   for(int i = 0; i < n; i++)cout << q[i].f() << " ";
   cout<<"\n";
}

int main()
{
   B* p = new B[2];func(p, 2); delete[] p;
   p = new D; func(p, 1); delete p;
   cout << D::get_x();
}
//Nu compileaza 0.1
//Nu putem apela o metoda non-statica intr-o metoda statica
//(nu exista *this) 0.2
//Pentru a compila trebuie sa scoatem identificatorul static pentru
//metoda f() din clasa B 0.2
```

Exercitiul 6

var 1

```
#include <iostream>
using namespace std;

class A{
   int x;
public:
   A(int i = 25){x = i; }
   int& get_x() const { return x; }
   void set_x(int i) {  x = i; }
   A& operator=(A a1){
      set_x(a1.get_x());
      return *this;
   }
};

int main()
{
   A a(18), b(7);
   (b=a).set_x(27);
   int i;
   i=b.get_x();
   return 0;
}
//Nu compileaza   0.1p
//Nu putem returna o referinta dintr-o functie constanta
//(s-ar putea modifica ulterior valoarea respectiva dupa get_x())  0.2p
//Pentru a compila fie scoatem identificatorul const
//fie nu mai returnam int& ci returnam doar int (scoatem &) 0.2p
```

---

var 2

```
#include <iostream>
using namespace std;

class A{
   int x;
public:
   A(int i = 25){ x = i; }
   int get_x() const{ return x;}
   void set_x(int i){  x=i; }
```

```
    A operator=(A a1)
    {
        set_x(a1.get_x());
        return a1;
    }
};

int main()
{
    A a(18), b(7);
    (b=a).set_x(27);
    int i;
    i=b.get_x();
    return 0;
}
//Compileaza si da valoarea 18 lui i
//  0.1 pentru ca copileaza si ceva explicatii
//  0.2 p pentru 27 si ajunge la 0.3 p
// 0.4 p pentru 18  si ajunge la 0.5 p
// alte valori (7, 25) 0.1 si ajunge la 0.2 p
```

---

var 3

```
#include <iostream>
using namespace std;

class A
{
    int *x;
public:
    A(int i = 25){ x = new int(i); }
    int& get_x() const  { return *x;  }
    void set_x(int i) { x = new int(i); }
    A& operator=(A a1)
    {
        set_x(a1.get_x());
        return *this;
    }
};

int main()
{
    A a(18), b(7);
    (b=a).set_x(27);
    int i;
```

```
   i=b.get_x();
   return 0;
}
//Compileaza si se afiseaza 27
//0.1p ca compileaza si alte explicatii
// 0.2p pentru explicatia de ce b va avea valoarea 27 si nu 18 functia = intaorce referinta etc
// 0.2p pentru 18
// daca zice ca afiseaza 18 are 0.3p
// daca zice alte valori apropiate: 25, 7 0.2p
```

---

var 4

```
#include <iostream>
using namespace std;

class A {
   int *x;
public:
   A() {  x = new int(0);  }
   A(int i) { x = new int(i); }
   int& get_x() const { return *x; }
   void set_x(int i) { x = new int(i); }
   A operator=(A a1) { set_x(a1.get_x());return a1;}
};

class B : public A {
   int y;
public:
   B() : A() { y = 0; }
   B(int i) : A(i){ y = i;}
   void afisare() const { cout << y; }
};

int main()
{
   B a(112), b, *c;
   int i;
   i=(b = a).get_x();
   (c = &a)->afisare();
   return 0;
}

//Compileaza si afiseaza
//112 si i va avea valoarea 112
```

// 0.1 pentru ca compileaza si descrierea variabilelor
// 0.2p pentru justificarea ca i ia valoarea 112
// 0.2 p pentru afisarea pe ecran a lui 112

---

var 5

```
#include <iostream>
using namespace std;

class A {
   int x;
public:
   A(){  x = 0; }
   A(int i) { x = i; }
   int& get_x() const { return x; }
   void set_x(int i) { x = i; }
   A operator=(A a1) { set_x(a1.get_x()); return a1;}
};

class B : public A {
   int y;

public:
   B() : A(){ y = 0;}
   B(int i) : A(i) { y = i; }
   void afisare() const { cout << y; }
};

int main()
{
   B a(112), b, *c;
   int i;
   i= (b = a).get_x()<<"\n";
   (c = &a)->afisare();
   return 0;
}

//Nu compileaza  0.1p
//Nu putem returna o referinta dintr-o functie constanta
//(s-ar putea modifica ulterior valoarea respectiva dupa get_x())  0.2p
//Pentru a compila fie scoatem identificatorul const
//fie nu mai returnam int& ci returnam doar int (scoatem &)        0.2p
```

Exercitiul 7

Variantele unde compileaza (var 1, 2 si 5):
0.1p raspunsul corect.
0.1 – explicatii var locala x din functie
0.2 – explicatii constructor obiect 2
0.1 – explicatii x extern

Variantele unde nu compileaza:
0p - Nu compileaza si atat
0.1p - Nu compileaza si motiv gresit
0.3p - Nu compileaza si motiv corect
0.2p - Modificarea

var 1 - 20 30 40 40 100

```
2   #include <iostream>
3   using namespace std;
4   int x = 10;
5   void f(int x)
6   {
7       cout<<x<<"  ";
8       class A {
9           protected:  int x;
10          public:
11          A(int a = 30){x = a; cout<<x<<"  ";}
12      } ;
13      class B: public A {
14          int x;
15          public:
16          B(int b = 40){x = b;  cout<<x<<"  "<<x<<"  ";}
17          int afis() {return x + A::x;}
18      } ob2;
19      cout<<x + ::x + ob2.afis();
20  }
21
22  int main()
23  {
24      f(20);
25  }
```

var 2 – 20 30 20 20 70

```
#include <iostream>
using namespace std;
int x = 10;
void f()
{
    static int x = 20;
    cout<<x<<" ";
    class A {
       protected: int x;
       public:
       A(int a = 30){x = a; cout<<A::x<<" ";}
    } ;
    class B: public A {
       int x;
       public:
       B(int b = 20){x = b; cout<<x<<" "<<x<<" ";}
       int afis() {return ::x + A::x;}
    } ob2;
    cout<<x + ::x + ob2.afis();
}

int main()
{
    f();
}
```

var 3 – nu compileaza pentru ca intr-o clasa locala functiile nu pot fi definite in afara clasei

```
#include <iostream>
using namespace std;
int x = 10;
void f(int x)
{
    cout<<x<<" ";
    class A {
       protected: int x;
       public:
       A(int a = 20){x = a; cout<<x<<" ";}
    } ;
    class B: public A {
       int x;
       public:
       B(int b = 30){x = b; cout<<x<<" "<<x<<" ";}
       int afis();
    } ob2;
    int B::afis() {return x + A::x;}
    cout<<x + ::x + ob2.afis();
}

int main()
{
    f(40);
}
```

var 4 – nu compileaza pentru ca o clasa locala nu poate avea variabile statice

```
#include <iostream>
using namespace std;
int x = 50;
void f(int x)
{
    cout<<x<<"  ";
    class A {
        protected:  int x;
        public:
        A(int a = 20){x = a; cout<<x<<"  ";}
    } ;
    class B: public A {
        static int x;
        public:
        B(){cout<<x<<"  ";}
        int afis(){return x + A::x;}
    } ob;
    int B::x = 30;
    cout<<x + ::x + ob.afis();
}

int main()
{
    f(40);
}
```

var 5 – compileaza pt ca se accepta functii statice in clase locale: 40 20 30 60

```
#include <iostream>
using namespace std;
int x = 30;
void f(int x)
{
    cout<<x<<"  ";
    class A {
        protected:  int x;
        public:
        A(int a = 20){x = a; cout<<x<<"  ";}
    } ;
    class B: public A {
        int x;
        public:
        B(int b = 30){x = b; cout<<x<<"  ";}
        static int afis(){return ::x;}
    } ob;
    cout<<::x + B::afis();
}

int main()
{
    f(40);
}
```

Exercitiul 8

Variantele unde compileaza (var 2 si 3):
0.1p raspunsul corect.
0.2 puncte explicatii pentru prima parte afisata.
0.2 puncte pentru cea de-a doua.

Variantele unde nu compileaza:
0p - Nu compileaza si atat
0.1p - Nu compileaza si motiv gresit
0.3p - Nu compileaza si motiv corect
0.2p - Modificarea

var 1 - nu compileaza pt ca in clasa nested nu putem accesa direct data nestatica din clasa mare

```
2   #include<iostream>
3   using namespace std;
4
5   class X
6   {
7       int x;
8       class Y
9       {
10          int y;
11          public:
12          Y(int b = 2){y = b; cout<<y<<"  ";}
13          void afis(){cout<<x<< "  "<<y;}
14      }ob1;
15      public:
16      X(int a = 1){x = a; cout<<x<<"  ";}
17      void afis(){ ob1.afis();}
18  };
19
20  int main()
21  {
22      X ob2;
23      ob2.afis();
24  }
```

var 2 – 2 1 1 2

```
#include<iostream>
using namespace std;

class X
{
    static int x;
    class Y
    {
        int y;
        public:
        Y(int b = 2){y = b; cout<<y<<"  ";}
        void afis(){cout<<x<< "  "<<y;}
    }ob1;
    public:
    X(int a = 1){x = a; cout<<x<<"  ";}
    void afis(){ ob1.afis();}
};
int X::x = 3;
int main()
{
    X ob2;
    ob2.afis();
}
```

var 3 – 1 2 1 10 10 – se pot accesa variabile private ale obiectelor din clasa mare

```
#include<iostream>
using namespace std;

class X
{
    int x;
    class Y
    {
        int y;
        public:
        Y(int b = 1){y = b; cout<<y<<"  ";}
        void afis(){X ob3(10); cout<<ob3.x;}
    }ob1;
    public:
    X(int a = 2){x = a; cout<<x<<"  ";}
    void afis(){ ob1.afis();}
};
int main()
{
    X ob2;
    ob2.afis();
}
```

var 4 – ob1 e privat – nu compileaza

```
2   #include<iostream>
3   using namespace std;
4
5   class X
6   {
7       int x;
8       class Y
9       {
10          int y;
11          public:
12          Y(int b = 1){y = b; cout<<y<<"  ";}
13          void afis(){X ob3(4); cout<<ob3.x;}
14      }ob1;
15      public:
16      X(int a = 2){x = a; cout<<x<<"  ";}
17      void afis(){ ob1.afis();}
18  };
19  int main()
20  {
21      X ob2;
22      ob2.ob1.afis();
23  }
```

var 5 – nu compileaza – Y nu e vizibila din main

```
2   #include<iostream>
3   using namespace std;
4
5   class X
6   {
7       int x;
8       class Y
9       {
10          int y;
11          public:
12          Y(int b = 1){y = b; cout<<y<<"  ";}
13          void afis(){X ob3(4); cout<<ob3.x;}
14      }ob1;
15      public:
16      X(int a = 2){x = a; cout<<x<<"  ";}
17      void afis(){ ob1.afis();}
18  };
19  int main()
20  {
21      Y ob2;
22      ob2.afis();
23  }
```

Exercitiul 9

Variantele unde compileaza (var 1, 2 si 5):

0.1p raspunsul corect.
0.2 – trimitere catre constructorul de initializare din B
0.1 x si y sunt 0 si I este 1,2,sau 3
0.1 – se apeleaza suma din derivata

x si y sunt 0, se apeleaza constr de init din baza

Variantele unde nu compileaza:
0p - Nu compileaza si atat
0.1p - Nu compileaza si motiv gresit
0.3p - Nu compileaza si motiv corect
0.2p - Modificarea

Varianta 3: nu compileaza, "." in loc de "->" in constructori
Varianta 4: nu compileaza, nu exista constructor neparametrizat in baza

Varianta 1: d.Suma() = 1.5

```
class Baza {
protected:
   int x,y;
public:
   Baza() {
      this->x = 0;
      this->y = 0;
   }
   Baza(int x, int y) {
      this->x = x;
      this->y = y;
   }
   int Suma() {return x + y;}
};

class Derivata : public Baza {
   double t;
public:
   Derivata(int x, int y, double t) {
      Baza(x,y);
      this -> t = t;
   }
   double Suma() {return x + y + t;}
};
```

```
int main() {
   Derivata d(5, 3, 1.5);
   int i= d.Suma();
}
```

Varianta 2: d.Suma() = 2.5

```
class Baza {
protected:
   int x,y;
public:
   Baza() {
      this->x = 0;
      this->y = 0;
   }
   Baza(int x, int y) {
      this->x = x;
      this->y = y;
   }
   int Suma() {return x + y;}
};

class Derivata : public Baza {
   double t;
public:
   Derivata() {
      this -> t = 2.5;
   }
   Derivata(Derivata& derivata) {
      Baza(derivata.x + 1, derivata.y + 1);
      this -> t = derivata.t;
   }
   double Suma() {return x + y + t;}
};

int main() {
   Derivata d;
   Derivata d1(d);
   int i=  d1.Suma();
}
```

Varianta 3: nu compileaza, "." in loc de "->" in constructori

```
class Baza {
protected:
   int x,y;
public:
   Baza() {
      this.x = 0;
      this.y = 0;
   }
   Baza(int x, int y) {
      this.x = x;
      this.y = y;
   }
   int Suma() {return x + y;}
};

class Derivata : public Baza {
   double t;
public:
   Derivata() {
      this.t = 2.5;
   }
   Derivata(Derivata& derivata) {
      Baza(derivata.x + 1, derivata.y + 1);
      this.t = derivata.t;
   }
   double Suma() {return x + y + t;}
};

int main() {
   Derivata d;
   Derivata d1(d);
   int i=  d1.Suma();
}
```

Varianta 4: nu compileaza, nu am constructor neparametrizat in baza

```
class Baza {
protected:
   int x,y;
public:
   Baza(int x, int y) {
      this->x = x;
      this->y = y;
```

```
    }
    int Suma() {return x + y;}
};

class Derivata : public Baza {
    double t;
public:
    Derivata() {
        this -> t = 2.5;
    }
    Derivata(Derivata& derivata) {
        Baza(derivata.x + 1, derivata.y + 1);
        this -> t = derivata.t;
    }
    double Suma() {return x + y + t;}
};

int main() {
    Derivata d;
    Derivata d1(d);
    int i=  d1.Suma();
}
```

Varianta 5: d.Suma() = 3.5

```
class Baza {
protected:
    int x,y;
public:
    Baza() {
        this->x = 0;
        this->y = 0;
    }
    Baza(int x, int y) {
        this->x = x;
        this->y = y;
    }
    int Suma() {return x + y;}
};

class Derivata : public Baza {
    double t;
public:
    Derivata() {
        Baza(1,1);
```

```
        this -> t = 3.5;
    }
    Derivata(Derivata& derivata) {
        Baza(derivata.x + 1, derivata.y + 1);
        this -> t = derivata.t;
    }
    double Suma() {return x + y + t;}
};

int main() {
    Derivata d;
    Derivata d1(d);
    int i=  d1.Suma();
}
```

Exercitiul 10

```
////////////////////////////////////
////   1
////////////////////////////////////
#include <iostream>
using namespace std;
class C {
   int const *p;
   public:
      C (int *q) : p(q) {}
      void reload () { delete p; p = new int;}
      void set (const int * const q) { *p = *q; }
};
int main () {
   int x = 20210614;
   C ob(&x);
   const int& rx = x;
   ob.reload(); ob.set(&rx);
   return 0;
}


// Secventa nu compileaza din cauza liniei 9 deoarece p este un pointer catre constanta (isi poate
modifica adresa, nu poate modifica valoarea salvata la aceea adresa)
// Rezolvare:
//    modificarea liniei 5 in: int *p;

// Barem:
// Total: 0.5p
// 1.a 0.1p secventa nu compileaza cu explicatii care nu sunt corecte
// 1.b 0.25p secventa nu compileaza cu explicatii corecte
// 2 0.25p Rezolvarea problemei de compilare prin solutia indicata
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.
// OBS! Punctajul de punctul 2 se aloca doar daca eroare la compilare este identificata corect.


////////////////////////////////////
////   2
////////////////////////////////////

#include <iostream>
using namespace std;
class C {
   int *p;
   public:
      C (int *q) : p(q) {}
```

```
        void reload () { delete p; p = new int;}
        void set (int q) const { *p = q; }
};
int main () {
    int x = 20210614;
    C ob(&x);
    ob.reload(); ob.set(x);
    return 0;
}
```

// Secventa compileaza si afiseaze eroare la runtime deoarece se incearca dezalocarea unei adrese care nu a fost alocata dinamic:

// main(57571,0x11b544e00) malloc: *** error for object 0x7ffee2ab15b8: pointer being freed was not allocated
// main(57571,0x11b544e00) malloc: *** set a breakpoint in malloc_error_break to debug
// Abort trap: 6

// Barem:
// Total: 0.5p
// 1.a 0.1p secventa compileaza si afiseaza orice mai putin raspunsul corect
// 1.b 0.25p secventa copileaza si afiseaza exceptia la dezalocare
// 2 0.25p explicare comportamentului secventei de cod (daca explicatiile sunt partial corecte, ~50%, se acorda jumatate din punctaj)
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.

//////////////////////////////////////
//// 3
//////////////////////////////////////

```
#include <iostream>
using namespace std;
class C {
    int * p;
    public:
        C (int *q) : p(q) {}
        void reload () { delete p; p = new int;}
        void set (const int * const q) { *p = *q + 13; }
        operator int () {return *p;}
};
int main () {
    int *x = new int(20210601); const int& rx = *x;
    C ob(x);
    ob.reload(); ob.set(&rx);
    cout << ob;
```

```
    return 0;
}

// Secventa compileaza si afiseaze 20210614
// Barem:
// Total: 0.5p
// 1.a 0.1p secventa compileaza si afiseaza un numar apropiat de 20210614, dar nu 20210614
// 1.b 0.25p secventa compileaza si afiseaza 20210614
// 2 0.25p explicare comportamentului secventei de cod (daca explicatiile sunt partial corecte,
~50%, se acorda jumatate din punctaj)
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.

////////////////////////////////////
////   4
////////////////////////////////////

#include <iostream>
using namespace std;
class C {
    int * const p;
    public:
        C (int q) : p(new int(q)) {}
        void set (const int& q) const { *p = q + 86; }
        operator int () const {return *p;}
};
int main () {
    const C ob(91973549);
    ob.set(422032);
    cout << ob;
    return 0;
}

// Secventa compileaza si afiseaze 422118
// Barem:
// Total: 0.5p
// 1.a 0.1p secventa compileaza si afiseaza un numar apropiat de 422118, dar nu 422118
// 1.b 0.25p secventa compileaza si afiseaza 422118
// 2 0.25p explicare comportamentului secventei de cod (daca explicatiile sunt partial corecte,
~50%, se acorda jumatate din punctaj)
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.

////////////////////////////////////
////   5
////////////////////////////////////

#include <iostream>
```

```
using namespace std;
class C {
   int * const p;
   public:
      C (int q) : p(new int(q)) {}
      void set (const int& q) const { *p = q + 59; }
      operator int () {return *p;}
};
int main () {
   const C ob(22973890);
   ob.set(488474);
   cout << ob;
   return 0;
}


// Secventa nu compileaza din cauzaa liniei 14, deoarece operatorul << nu este supraincarcat pt clasa C, iar operatorul de conversie la int nu poate fi folosit deoarece obiectul ob este constant
// Rezolvare:
//     modificare liniei 9 in: operator int () const {return *p;}


// Barem:
// Total: 0.5p
// 1.a 0.1p secventa nu compileaza cu explicatii care nu sunt corecte
// 1.b 0.25p secventa nu compileaza cu explicatii corecte
// 2 0.25p Rezolvarea problemei de compilare prin solutia indicata
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.
// OBS! Punctajul de punctul 2 se aloca doar daca eroare la compilare este identificata corect.
```

Exercitiul 11

```
////////////////////////////////////
////   1
////////////////////////////////////
#include <iostream>
using namespace std;
class Vector2D {
   int x, y;
public:
   Vector2D(const int& a, const int& b) : x(a), y(b) {}
   Vector2D operator+ (Vector2D v) {
      Vector2D w(x + v.x, y + v.y); return w;
   }
   friend ostream& operator<< (ostream& out, Vector2D& v) {
      out << "(" << v.x << ", " << v.y << ")"; return out;
   }
};
int main () {
   cout << Vector2D(10, 5) + Vector2D(22, 159);
   return 0;
}

// Secventa nu compileaza din cauza liniei 16 deoarece operatorul << poate fi apelat doar cu obiected de tip Vector2D care au adresa in memorie (i.e. non constante)
// Rezolvare:
// 1 - modificare linia 12 in: friend ostream& operator<< (ostream& out, const Vector2D& v) {
// 2 - modificare linia 12 in: friend ostream& operator<< (ostream& out, Vector2D v) {

// Barem:
// Total: 0.5p
// 1.a 0.1p secventa nu compileaza cu explicatii care nu sunt corecte dar in zona supraincari de operator
// 1.b 0.25p secventa nu compileaza cu identificare corecta a problemei conform explicatiei de mai sus
// 2 0.25p rezolvarea problemei prin una din solutiile indicate
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.
// OBS! Punctajul de punctul 2 se aloca doar daca eroare la compilare este identificata corect.

////////////////////////////////////
////   2
////////////////////////////////////

#include <iostream>
using namespace std;
class Vector2D {
   int x, y;
```

```
public:
   Vector2D(const int& a, const int& b) : x(a), y(b) {}
   Vector2D operator+ (Vector2D v) {
      Vector2D w(x + v.x, y + v.y); return w;
   }
   friend ostream& operator<< (ostream& out, const Vector2D& v) {
      out << "(" << v.x << ", " << v.y << ")"; return out;
   }
};
int main () {
   cout << Vector2D(2, 5) + Vector2D(4, 8);
   return 0;
}

// Secventa compileaza si afiseaza (6, 13)

// Barem:
// Total: 0.5p
// 1.a 0.1p secventa compileaza si afiseaza (x, y) - i.e orice 2 numere intregi
// 1.b 0.25p secventa compileaza si afiseaza (6, 13)
// 2 0.25p explicare comportamentului secventei de cod (daca explicatiile sunt partial corecte,
~50%, se acorda jumatate din punctaj)
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.

//////////////////////////////////////
////   3
//////////////////////////////////////

#include <iostream>
using namespace std;
class Vector2D {
   int x, y;
public:
   Vector2D(const int& a, const int& b) : x(a), y(b) {}
   friend Vector2D operator+ (Vector2D& v, Vector2D& u) {
      Vector2D w(u.x + v.x, u.y + v.y); return w;
   }
   friend ostream& operator<< (ostream& out, const Vector2D& v) {
      out << "(" << v.x << ", " << v.y << ")"; return out;
   }
};
int main () {
   cout << Vector2D(45, 29) + Vector2D(87, 10);
   return 0;
}
```

// Secventa nu compileaza din cauza liniei 16 deoarece operatorul + se asteapta la 2 obiecte Vector2D cu adresa in memorie, iar obiectele anonime sunt obiecte constante
// Rezolvare
// 1. modificarea liniei 8 in: friend Vector2D operator+ (const Vector2D& v, const Vector2D& u) {
// 2. modificarea liniei 8 in: friend Vector2D operator+ (Vector2D v, Vector2D u) {

// Barem:
// Total: 0.5p
// 1.a 0.1p secventa nu compileaza cu explicatii care nu sunt corecte
// 1.b 0.25p secventa nu compileaza cu explicatii corecte
// 2 0.25p Rezolvarea problemei de compilare prin solutia indicata
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.
// OBS! Punctajul de punctul 2 se aloca doar daca eroare la compilare este identificata corect.

////////////////////////////////////
//// 4
////////////////////////////////////

```
#include <iostream>
using namespace std;
class Vector2D {
   int x, y;
   const int translate = 10 ;
public:
   Vector2D(const int& a, const int& b) : x(a), y(b) {}
   Vector2D operator* (const Vector2D& v) {
      return Vector2D(translate + x * v.x, translate + y + v.y);
   }
   friend ostream& operator<< (ostream& out, const Vector2D& v) {
      out << "(" << v.x << ", " << v.y << ")"; return out;
   }
};
int main () {
   cout << Vector2D(98, 69) * Vector2D(82, 12);
   return 0;
}
```

// Secventa compileaza si afiseaza (8046, 91)

// Barem:
// Total: 0.5p
// 1.a 0.1p secventa compileaza si afiseaza (x, y) - i.e orice 2 numere intregi
// 1.b 0.25p secventa compileaza si afiseaza (8046, 91)
// 2 0.25p explicare comportamentului secventei de cod (daca explicatiile sunt partial corecte, ~50%, se acorda jumatate din punctaj)

// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.

////////////////////////////////////
//// 5
////////////////////////////////////

```
#include <iostream>
using namespace std;
class Vector2D {
   int x, y;
public:
   Vector2D(const int& a, const int& b) : x(a), y(b) {}
   Vector2D operator* (Vector2D v) const {
      return Vector2D(x + v.y, y + v.x);
   }
   friend ostream& operator<< (ostream& out, const Vector2D& v) {
      out << "(" << v.x << ", " << v.y << ")"; return out;
   }
};
int main () {
   cout << Vector2D(33, 40).operator*(Vector2D(77, 60));
   return 0;
}
```

// Secventa compileaza si afiseaza (93, 117)

// Barem:
// Total: 0.5p
// 1.a 0.1p secventa compileaza si afiseaza (x, y) - i.e orice 2 numere intregi
// 1.b 0.25p secventa compileaza si afiseaza (93, 117)
// 2 0.25p explicare comportamentului secventei de cod (daca explicatiile sunt partial corecte, ~50%, se acorda jumatate din punctaj)
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.

Exercitiul 12

```
////////////////////////////////////
////   1
////////////////////////////////////
#include <iostream>
using namespace std;
class A {
public:
   A () {cout << "A";}
   ~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 () {
   A *pa = new C(); delete pa;
   return 0;
}

// Secventa compileaza si afiseaza ABC~A

// Barem:
// Total: 0.5p
// 1.a 0.1p secventa compileaza si afiseaza ABC~C~B~A sau ABC~B~A
// 1.b 0.25p secventa compileaza si afiseaza ABC~A
// 2 0.25p explicare comportamentului secventei de cod (daca explicatiile sunt partial corecte,
~50%, se acorda jumatate din punctaj)
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.

////////////////////////////////////
////   2
////////////////////////////////////

#include <iostream>
using namespace std;
class A {
public:
   A () {cout << "A";}
   ~A () {cout << "~A";}
```

```
};
class B: A {
public:
   B () {cout << "B";}
   ~B () {cout << "~B";}
};
class C: public B {
public:
   C () {cout << "C";}
   ~C () {cout << "~C";}
};
int main () {
   A *pa = new C(); delete pa;
   return 0;
}
```

// Secventa nu compileaza din cauza liniei 20 deoarece A este o baza privata a lui C, deci nu se poate face upcasting

// Barem:
// Total: 0.5p
// 1.a 0.1p secventa nu compileaza cu explicatii care nu sunt corecte
// 1.b 0.25p secventa nu compileaza cu explicatii corecte
// 2 0.25p Rezolvarea problemei de compilare prin modificarea liniei 8 in: class B: public A {
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.
// OBS! Punctajul de punctul 2 se aloca doar daca eroare la compilare este identificata corect.

////////////////////////////////////
//// 3
////////////////////////////////////

```
#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 () {
   A *pa = new C(); delete pa;
   return 0;
}

// Secventa compileaza si afiseaza ABC~C~B~A

// Barem:
// Total: 0.5p
// 1.a 0.1p secventa compileaza si afiseaza ABC~B~A sau ABC~A
// 1.b 0.25p secventa compileaza si afiseaza ABC~C~B~C
// 2 0.25p explicare comportamentului secventei de cod (daca explicatiile sunt partial corecte, ~50%, se acorda jumatate din punctaj)
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.

////////////////////////////////////
////   4
////////////////////////////////////

#include <iostream>
using namespace std;
class A {
public:
   A () {cout << "A";}
   ~A () {cout << "~A";}
};
class B: public A {
public:
   B () {cout << "B";}
   ~B () {cout << "~B";}
};
class C: public A, public B {
public:
   C () {cout << "C";}
   ~C () {cout << "~C";}
};
int main () {
   A *pa = new C(); delete pa;
   return 0;
}

// Secventa nu compileaza din cauza liniei 20 deoarece A este o baza ambigua pentru C si compilatorul nu stie pe care sa o foloseasca pentru upcasting
```

// Barem:
// Total: 0.5p
// 1.a 0.1p secventa nu compileaza cu explicatii care nu sunt corecte
// 1.b 0.25p secventa nu compileaza cu explicatii corecte
// 2 0.25p Rezolvarea problemei de compilare prin modificarea liniei 13 in: class C: public B {
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.
// OBS! Punctajul de punctul 2 se aloca doar daca eroare la compilare este identificata corect.

////////////////////////////////////
//// 5
////////////////////////////////////

```
#include <iostream>
using namespace std;
class A {
public:
   A () {cout << "A";}
   ~A () {cout << "~A";}
};
class B: public A {
public:
   B () {cout << "B";}
   virtual ~B () {cout << "~B";}
};
class C: public A, public B {
public:
   C () {cout << "C";}
   ~C () {cout << "~C";}
};
int main () {
   B *pb = new C(); delete pb;
   return 0;
}
```

// Secventa compileaza si afiseaza AABC~C~B~A~A

// Barem:
// Total: 0.5p
// 1.a 0.1p secventa compileaza si afiseaza ABC~C~B~A sau ABC~B~A sau ABC~A
// 1.b 0.25p secventa compileaza si afiseaza AABC~C~B~A~A
// 2 0.25p explicare comportamentului secventei de cod (daca explicatiile sunt partial corecte, ~50%, se acorda jumatate din punctaj)
// OBS! Doar una dintre 1.a sau 1.b poate fi alocata.