Problema 5:

Barem:

Variantele 1,2:

Nu si motivatie corecta + modificari corecte: 0.5
Nu si motivatie corecta, dar fara modificari: 0.3
Nu si motivatie aproape corecta: 0.1

Variantele 3, 4,5:

Da si afisarile corecte, cu explicatii complete si coerente (despre constructori + apel de functie): 0.5
Da si la explicatii nu se spune nimic despre apelul functiei: 0.3
Da si rezultat aproape corect, cu explicatii: 0.1

Alte penalizari la explicatii:

- nu se explica de ce se merge pe functia din A desi se apeleaza un obiect de tip B: penalizare 0.1
- nu se mentioneaza ascunderea functiei fara parametri de catre cea cu parametru (supraincarcarea functiilor virtuale): penalizare 0.1

Da sau nu fara explicatii nu se puncteaza.

Varianta 1: NU COMPILEAZA linia 21 da eroare: am.r(8) trebuia fara parametru

```
#include <iostream>

class A {
protected:
    int nm;
public:
    A(int hbr = 1) : nm(hbr) { std::cout << "?"; }
    int ha() { return nm; }
    virtual void r() const {};
    virtual ~A() {};
};

class B : public A {
    int d;
public:
    B(int b = 2) : d(b) { std::cout << "!!"; }
    void r(int z) const  { std::cout << nm << " " << z << "\n"; }
};

void warranty(const A& am) {
    am.r(8);
}
```

```
int main() {
    A ha;
    B un(ha.ha());
    warranty(un);
}
```

Varianta 2: NU COMPILEAZA se incearca instantierea unei clase abstracte

```
#include <iostream>

class A {
protected:
    int nm;
public:
    A(int hbr = 1) : nm(hbr) { std::cout << "?"; }
    int ha() { return nm; }
    virtual void r() const {};
    virtual ~A()=0;
};

class B : public A {
    int d;
public:
    B(int b = 2) : d(b) { std::cout << "!!"; }
    void r(int z) const  { std::cout << nm << " " << z << "\n"; }
};

void warranty(const A& am) {
    am.r();
}

int main() {
    A ha;
    B un(ha.ha());
    warranty(un);
}
```

Varianta 3: COMPILEAZA si afiseaza ??!!

```
#include <iostream>

class A {
protected:
    int nm;
public:
    A(int hbr = 1) : nm(hbr) { std::cout << "?"; }
    int ha() { return nm; }
    virtual void r() const {};
    virtual ~A() {};
};

class B : public A {
    int d;
public:
    B(int b = 2) : d(b) { std::cout << "!!"; }
```

```
    void r(int z) const  { std::cout << nm << " " << z << "\n"; }
};

void warranty(const A& am) {
    am.r();
}

int main() {
    A ha;
    B un(ha.ha());
    warranty(un);
}
```

Varianta 4: COMPILEAZA si afiseaza ????!

```
#include <iostream>

class A {
protected:
    int nm;
public:
    A(int hbr = 1) : nm(hbr) { std::cout << "??"; }
    int ha() { return nm; }
    virtual void r() const {};
    virtual ~A() {};
};

class B : public A {
    int d;
public:
    B(int b = 2) : d(b) { std::cout << "!"; }
    void r(int z) const  { std::cout << nm << " " << z << "\n"; }
};

void warranty(const A& am) {
    am.r();
}

int main() {
    A ha;
    B un(ha.ha());
    warranty(un);
}
```

Varianta 5: COMPILEAZA si afiseaza ????! 1 8

```
#include <iostream>

class A {
protected:
    int nm;
public:
    A(int hbr = 1) : nm(hbr) { std::cout << "??"; }
```

```
    int ha() { return nm; }
    virtual void r() const {};
    virtual ~A() {};
};

class B : public A {
    int d;
public:
    B(int b = 2) : d(b) { std::cout << "!"; }
    void r(int z) const  { std::cout << nm << " " << z << "\n"; }
};

void warranty(const B& am) {
    am.r(8);
}

int main() {
    A ha;
    B un(ha.ha());
    warranty(un);
}
```

Problema 6:

Barem:

Variantele 1,2:

Nu si motivatie corecta + modificari corecte: 0.5
Nu si motivatie corecta, dar fara modificari: 0.3
Nu si motivatie aproape corecta: 0.1

Variantele 3, 4,5:

Da si afisarile corecte, cu explicatii complete si coerente: 0.5
Da si raspunsul corect, fara explicatii si de ce : 0.25
Da si rezultat aproape corect, cu explicatii: 0.1

Alte penalizari la explicatii:
o<<a+&a nu rezolva problema (nu mai are eroare la compilare, dar are la executie. Cea mai rapida: o<<a.getx()+a.getx() – penalizare 0.1

Da sau nu fara explicatii nu se puncteaza.

Varianta 1: NU COMPILEAZA linia 11 da eroare: x e private

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

class A
{
    int x;
public:
    A(int y=0):x(y) {}
    int getx()
    {
        return x;
    };
    A operator+( A* o)
    {
        A r(*this);
        r.x=r.x+o->x-1;
        return r;
    }
};
ostream& operator<<(ostream& o, A a)
{
    o<<a.x;
    return o;
}
int main()
{
```

```
    A o1(101);
    cout <<(o1+&o1);
    return 0;
}
```

Varianta 2: NU COMPILEAZA datorita a+a (nu exista op+ intre obiect si obiect

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

class A
{
    int x;
public:
    A(int y=0):x(y) {}
    int getx()
    {
        return x;
    };
    A operator+( A* o)
    {
        A r(*this);
        r.x=r.x+o->x+1;
        return r;
    }
};
ostream& operator<<(ostream& o, A a)
{
    o<<a+a;
    return o;
}
int main()
{
    A o1(101);
    cout <<(o1+&o1);
    return 0;
}
```

Varianta 3: COMPILEAZA si afiseaza 202

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

class A
{
    int x;
public:
    A(int y=0):x(y) {}
    int getx()
    {
        return x;
    };
    A operator+( A* o)
    {
        A r(*this);
        r.x=r.x+o->x;
```

```
        return r;
    }
};
ostream& operator<<(ostream& o, A a)
{
    o<<a.getx();
    return o;
}
int main()
{
    A o1(101);
    cout <<(o1+&o1);
    return 0;
}
```

Varianta 4: COMPILEAZA si afiseaza 205

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

class A
{
    int x;
public:
    A(int y=0):x(y) {}
    int getx()
    {
        return x;
    };
    A operator+( A* o)
    {
        A r(*this);
        r.x=r.x+o->x-1;
        return r;
    }
};
ostream& operator<<(ostream& o, A a)
{
    o<<a.getx();
    return o;
}
int main()
{
    A o1(103);
    cout <<(o1+&o1);
    return 0;
}
```

Varianta 5: COMPILEAZA si afiseaza 207

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

class A
{
    int x;
public:
```

```
    A(int y=0):x(y) {}
    int getx()
    {
        return x;
    };
    A operator+( A* o)
    {
        A r(*this);
        r.x=r.x+o->x+1;
        return r;
    }
};
ostream& operator<<(ostream& o, A a)
{
    o<<a.getx();
    return o;
}
int main()
{
    A o1(103);
    cout <<(o1+&o1);
    return 0;
}
```

Problema 7:

Barem:

Variantele 1,2:

Nu si motivatie corecta + modificari corecte: 0.5
Nu si motivatie corecta, dar fara modificari: 0.3
Nu si motivatie aproape corecta: 0.1

Variantele 3, 4,5:

Da si afisarile corecte, cu explicatii complete si coerente: 0.5
Da si raspunsul corect, fara explicatii si de ce : 0.25
Da si rezultat aproape corect, cu explicatii: 0.1

Alte penalizari la explicatii:
Stergerea in main nu e modificare. Corect operator+(int) – penalizare 0.1.
In general, daca nu e modificare care se leaga de POO – penalizare 0.1

Da sau nu fara explicatii nu se puncteaza.

Varianta 1: NU COMPILEAZA linia 28 a(7) da eroare: constructorul din A cere referinta; rezolvare A(int)

```
#include <iostream>
using namespace std;
class A
{
    int x;
public:
    A(int &i):x(i) {}
    int get_x()
    {
        return x;
    }
    A operator+(int&);
};
ostream& operator<<(ostream& o, A a)
{
    o<<a.get_x();
    return o;
}
A A::operator+(int &i)
{
    A t(i);
    return t;
}
int main()
{
```

```
    int b=77, c=9;
    A a(7);
    cout<<a+b+c<<" "<<a+c;
    return 0;
}
```

Varianta 2: NU COMPILEAZA datorita `a+7+c; rezolvare +(int)`

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

class A
{
    int x;
public:
    A(int &i):x(i) {}
    int get_x()
    {
        return x;
    }
    A operator+(int&);
};
ostream& operator<<(ostream& o, A a)
{
    o<<a.get_x();
    return o;
}
A A::operator+(int &i)
{
    A t(i);
    return t;
}
int main()
{
    int b=77, c=9;
    A a(b);
    cout<<a+7+c<<" "<<a+c;
    return 0;
}
```

Varianta 3: NU COMPILEAZA datorita a+7; rezolvare +(int)

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

class A
{
    int x;
public:
    A(int &i):x(i) {}
    int get_x()
    {
        return x;
    }
```

```
    A operator+(int&);
};
ostream& operator<<(ostream& o, A a)
{
    o<<a.get_x();
    return o;
}
A A::operator+(int &i)
{
    A t(i);
    return t;
}
int main()
{
    int b=77, c=9;
    A a(b);
    cout<<a+b+c<<" "<<a+7;
    return 0;
}
```

Varianta 4: COMPILEAZA si afiseaza si afiseaza 9 9

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

class A
{
    int x;
public:
    A(int &i):x(i) {}
    int get_x()
    {
        return x;
    }
    A operator+(int&);
};
ostream& operator<<(ostream& o, A a)
{
    o<<a.get_x();
    return o;
}
A A::operator+(int &i)
{
    A t(i);
    return t;
}
int main()
{
    int b=77, c=9;
    A a(b);
    cout<<a+b+c<<" "<<a+c;
    return 0;
}
```

Varianta 5: COMPILEAZA si afiseaza 77 77

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

class A
{
   int x;
public:
   A(int &i):x(i) {}
   int get_x()
   {
      return x;
   }
   A operator+(int&);
};
ostream& operator<<(ostream& o, A a)
{
   o<<a.get_x();
   return o;
}
A A::operator+(int &i)
{
   A t(i);
   return t;
}
int main()
{
   int b=77, c=9;
   A a(b);
   cout<<a+b<<" "<<a+c+b;
   return 0;
}
```

Problema 8:

Barem:

Variantele 1,2:

Nu si motivatie corecta + modificari corecte: 0.5
Nu si motivatie corecta, dar fara modificari: 0.3
Nu si motivatie aproape corecta: 0.1

Observatii:
Nu s-a dat efectiv declararea lui x sau s-a gresit linia la care sa se adauge 0.4
int A::X nu A int::x - 0.4

Variantele 3, 4,5:

Da si afisarile corecte, cu explicatii complete si coerente: 0.5
Da si raspunsul corect, fara explicatii si de ce : 0.25
Da si rezultat aproape corect, cu explicatii: 0.1

Alte penalizari la explicatii:
In general, daca nu e modificare care se leaga de POO – penalizare 0.1
Indicatii corecte, rezultat incorect - 0.3

Da sau nu fara explicatii nu se puncteaza.

Varianta 1: NU COMPILEAZA linia 5 da eroare: y trebuie initializat in lista de initializare

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

class A
{
    static int x;
    const int y;
public:
    A(int i)
    {
        x=i;
        y=-i+3;
    }
    int put_x(A a)
    {
        return x+a.y;
    }
};
int A::x=8;
int main()
{
```

```
    A a(2);
    cout<<a.put_x(26);
    return 0;
}
```

Varianta 2: NU COMPILEAZA linia 4 da eroare: static nedeclarat si in afara clasei

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

class A
{
    static int x;
    int y;
public:
    A(int i)
    {
        x=i;
        y=-i+3;
    }
    int put_x(A a)
    {
        return x+a.y;
    }
};
int main()
{
    A a(2);
    cout<<a.put_x(26);
    return 0;
}
```

Varianta 3: COMPILEAZA si afiseaza si afiseaza -23

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

class A
{
    static int x;
    const int y = 0;
public:
    A(int i)
    {
        x=i;
        x=-i+3;
    }
    int put_x(A a)
    {
        return x+a.y;
    }
};
int A::x=8;
```

```
int main()
{
    A a(2);
    cout<<a.put_x(26);
    return 0;
}
```

Varianta 4: COMPILEAZA si afiseaza si afiseaza 3

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

class A
{
   static int x;
   const int y = 10;
public:
   A(int i)
   {
      x=i;
      x=-y+3;
   }
   int put_x(A a)
   {
      return x+a.y;
   }
};
int A::x=8;
int main()
{
   A a(2);
   cout<<a.put_x(26);
   return 0;
}
```

Varianta 5: COMPILEAZA si afiseaza valoarea 23

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

class A
{
   static int x;
   const int y = 10;
public:
   A(int i)
   {
```

```
        x=i;
        x=y+3;
    }
    int put_x(A a)
    {
        return x+a.y;
    }
};
int A::x=8;
int main()
{
    A a(2);
    cout<<a.put_x(26);
    return 0;
}
```

Problema 9:

Varianta 1: NU COMPILEAZA linia 11 da eroare: pointerul p este constant deci nu poate apela o functie neconst

```
#include <iostream>
using namespace std;
class X
{ int i;
  public: X(int j=10)
       { i=j; cout<< i<< " ";}
       const int afisare(int j)
       { cout<<i<< " "; return i+j; }; };
int main()
{ const X O (7), &O2=O, *p=&O2;
  cout<<p->afisare(6);
  return 0; }
```

ca spune ca nu compileaza cu o justificare rezonabila 0.1p
ca linia 22: cout<<p->afisare(6); da eroare pointerul p este constant deci nu poate apela o functie neconst 0.2p

daca nu justifica bine modificarea nu conteaza (nu se puncteaza urmatoarea parte)

modificarea care il face sa mearga: afisare se defineste const la linia 13: 0.2p

Varianta 2: NU COMPILEAZA linia 11 da eroare: pointerul p este constant deci nu poate apela o functie neconst

```
#include <iostream>
using namespace std;
class X
{ int i;
  public: X(int j=10)
       { i=j; cout<< i<< " ";}
       int const afisare(int j)
       { cout<<i<< " "; return i+j; }; };
int main()
{ const X O (7), &O2=O, *p=&O2;
  cout<<p->afisare(6);
  return 0; }
```

ca spune ca nu compileaza cu o justificare rezonabila 0.1p
ca linia 22: cout<<p->afisare(6); da eroare pointerul p este constant deci nu poate apela o functie neconst 0.2p

daca nu justifica bine modificarea nu conteaza (nu se puncteaza urmatoarea parte)

modificarea care il face sa mearga: afisare se defineste const la linia 13: 0.2p

Varianta 3: COMPILEAZA si afiseaza si afiseaza valorile 1 1 7

```
#include <iostream>
using namespace std;
class X
{ int i;
  public: X(int j=10)
       { i=j; cout<< i<< " ";}
       int afisare(int j) const
       { cout<<i<< " "; return i+j; }; };
int main()
{ const X O (1), &O2=O, *p=&O2;
  cout<<p->afisare(6);
  return 0; }
```

spune ca compileaza 0.1p (daca spune doar ca compileaza nu primeste nimic) ca afiseaza 117 0.2p (daca spune ca compileaza si 1 1 7 fara altceva 0.1p)

Se apeleaza mai intai constructorul X cu parametrul 1 si se afiseaza 1 Apelam functia afisare prin pointer-ul p si vom afisa i-ul corespunzator pentru valoarea la care arata p, adica 1, iar apoi se intoarce spre cout adunarea 1 + 6 care are ca rezultat afisarea lui 7 pe ecran 0.2p

Varianta 4: COMPILEAZA si afiseaza si afiseaza valorile 7 7 13

```
#include <iostream>
using namespace std;
class X
{ int i;
  public: X(int j=10)
       { i=j; cout<< i<< " ";}
       int afisare(int j) const
       { cout<<i<< " "; return i+j; }; };
int main()
{ const X O (7), &O2=O, *p=&O2;
  cout<<p->afisare(6);
  return 0; }
```

spune ca compileaza 0.1p (daca spune doar ca compileaza nu primeste nimic) ca afiseaza 7 7 13 0.2p (daca spune ca compileaza si 7 7 13 fara altceva 0.1p)

Se apeleaza mai intai constructorul X cu parametrul 7 si se afiseaza 7 Apelam functia afisare prin pointer-ul p si vom afisa i-ul corespunzator pentru valoarea la care arata p, adica 7, iar apoi se intoarce spre cout adunarea 7 + 6 care are ca rezultat afisarea lui 13 pe ecran 0.2p

Varianta 5: COMPILEAZA si afiseaza valoarea 1 1 -5

```
#include <iostream>
using namespace std;
class X
{ int i;
  public: X(int j=10)
       { i=j; cout<< i<< " ";}
       int afisare(int j) const
       { cout<<i<< " "; return i-j; }; };
int main()
{ const X O (1), &O2=O, *p=&O2;
  cout<<p->afisare(6);
  return 0; }
```

spune ca compileaza 0.1p (daca spune doar ca compileaza nu primeste nimic)
ca afiseaza 11-5 0.2p (daca spune ca da si 1 1 -5 fara altceva 0.1p)

Se apeleaza mai intai constructorul X cu parametrul 1 si se afiseaza 1
Apelam functia afisare prin pointer-ul p si vom afisa i-ul corespunzator pentru valoarea la care arata p, adica 1, iar apoi se intoarce spre cout scaderea 1 - 6 care are ca rezultat afisarea lui -5 pe ecran 0.2p

Problema 10

Varianta care compileaza : da simplu 0p

-valori corecte: 0.3p

-valori apropriate (calcul matematic/semne greu vizibile) 0.2p

-valori indepartate 0.1p

-justificare (variabile si functii apelate) 0.2p

Varianta care nu compileaza: nu simplu/justificare gresita 0p

justificare 0.4p

modificare 0.1p

Varianta 1: NU COMPILEAZA linia 17 da eroare ca a nu e definit

```
#include <iostream>
using namespace std;
class A
{ int i;
  public: A(int x=8):i(x) {}
          virtual int f(A a)
          { return i+a.i; } };
class B: public A
{ int j;
  public: B(int x=-2):j(x) {}
          int f(B b) {return j+b.j; } };
int main()
{ int i=20;
  A *o;
  if (i%4) {A a; o=new A(i);}
  else {B b; o=new B(i);}
  cout<<a->f(*o);
  return 0; }
```

```
justificare :
- % ramura corecta (else) 0.1
-a e declarat in scopul  then 0.3
-a nu e pointer ci obiect 0.3
-modificare corecta 0.1
```

Varianta 2: NU COMPILEAZA -b nu e definit

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

```
      class A
      { int i;
        public: A(int x=8):i(x) {}
                virtual int f(A a)
                { return i+a.i; } };
      class B: public A
      { int j;
        public: B(int x=-2):j(x) {}
                int f(B b) {return j+b.j; } };
      int main()
      { int i=20;
        A *o;
        if (i%4) {A a; o=new A(i);}
        else {B b; o=new B(i);}
        cout<<b->f(*o);
        return 0; }
- % ramura corecta (else) 0.1
-b este creat doar in interiorul scopului else 0.3
-b nu este pointer 0.3
-modificare corecta 0.1
```

Varianta 3: COMPILEAZA si afiseaza valoarea 16

```
      #include <iostream>
      using namespace std;
      class A
      { int i;
        public: A(int x=8):i(x) {}
                virtual int f(A a)
                { return i+a.i; } };
      class B: public A
      { int j;
        public: B(int x=-2):j(x) {}
                int f(B b) {return j+b.j; } };
      int main()
      { int i=20;
        A *o;
        if (i%4) {A a; o=new A(i);}
        else {B b; o=new B(i);}
        cout<<o->f(*o);
        return 0; }
- % ramura corecta (else) 0.1p
- pt obiectul o -constructor cu param x=20 0.1p
-care apeleaza constructorul pt A cu valoarea implicita x=8 (sau subinteles
din calcul)0.1p
-apel f din clasa A -pt ca *o e la compilare vazut ca obiect din clasa A 0.1p
- rezultat  8+8=16 0.1p
```

Varianta 4: COMPILEAZA si afiseaza si afiseaza valoarea -16

```
      #include <iostream>
      using namespace std;
      class A
      { int i;
```

```
        public: A(int x=-8):i(x) {}
                virtual int f(A a)
                { return i+a.i; } };
      class B: public A
      { int j;
        public: B(int x=2):j(x) {}
                int f(B b) {return j+b.j; } };
      int main()
      { int i=20;
        A *o;
        if (i%4) {A a; o=new A(i);}
        else {B b; o=new B(i);}
        cout<<o->f(*o);
        return 0; }
- % ramura corecta (else) 0.1
- pt obiectul o -constructor cu param x=20 0.1p
-care apeleaza constructorul pt A cu valoarea implicita x=-8 (sau subinteles
din calcul)0.1p
-apel f din clasa A -pt ca *o e la compilare vazut ca obiect din clasa A 0.1p
- rezultat  -8-8=-16 0.1p
```

Varianta 5: COMPILEAZA si afiseaza valoarea 60

```
      #include <iostream>
      using namespace std;
      class A
      { int i;
        public: A(int x=-8):i(x) {}
                virtual int f(A a)
                { return i+a.i; } };
      class B: public A
      { int j;
        public: B(int x=2):j(x) {}
                int f(B b) {return j+b.j; } };
      int main()
      { int i=30;
        A *o;
        if (i%4) {A a; o=new A(i);}
        else {B b; o=new B(i);}
        cout<<o->f(*o);
        return 0; }
- % ramura corecta (then) 0.1
- pt obiectul o -constructor cu param x=30 0.1p
-apel f din clasa A  si  rezultat  30+30=60 0.3p
```

Problema 11:

Varianta 1: NU COMPILEAZA linia 10 da eroare

```
#include <iostream>
using namespace std;
template<class X>
X f(X x, X y)
{ return *x-*y; }
int f(double *x, int y)
{ return *(x+y); }
int main()
{ double a=10.7,*b=new double(21);
  cout<<f(a,b);
  return 0; }
-a e double, b e double* (sau subinteles) 0.1p
f(a,b)-nu gaseste potrivire
  - fara conversie la non-template 0.1p
  -fara conversie la template 0.1p
 - nu se poate face conversie la non-template 0.1p
 -varianta corecta 0.1p
```

Varianta 2: NU COMPILEAZA ca in operatorul << avem a.x si nu avem acces la x

```
#include <iostream>
using namespace std;
template<class X>
X f(X x, X y)
{ return *x-*y; }
int f(double *x, int y)
{ return *(x+y); }
int main()
{ double a=10.7,*b=new double(21);
  cout<<f(b,b);
  return 0; }

- fara conversie la non-template 0.1p
-fara conversie la template 0.1p
-rezultat *x -*y de tip double
- nu face conversia catre double * tipul intors de template 0.1
-varianta corecta 0.1p
```

Varianta 3: COMPILEAZA si afiseaza valoare nedetrminata

```
#include <iostream>
using namespace std;
template<class X>
X f(X x, X y)
{ return *x-*y; }
int f(double *x, int y)
{ return *(x+y); }
int main()
{ double a=10.7,*b=new double(21);
```

```
  cout<<f(b,a);
  return 0; }
- fara conversie la non-template 0.1p
-fara conversie la template 0.1p
-cu conversie la non template: a la int 0.1p
-x+y ca pointer 0.1p
_*(x+y) zona adresata nedeterminata 0.1p
```

Varianta 4: COMPILEAZA si afiseaza valoare nedetrminata

```
#include <iostream>

   using namespace std;
   template<class X>
   X f(X x, X y)
   { return *x-*y; }
   int f(double *x, int y)
   { return *(x+y); }
   int main()
   { double a=7.7,*b=new double(21);
     cout<<f(b,a);
     return 0; }

   - fara conversie la non-template 0.1p
   -fara conversie la template 0.1p
   -cu conversie la non template: a la int 0.1p
   -x+y ca pointer 0.1p
   _*(x+y) zona adresata nedeterminata 0.1p
```

Varianta 5: NU COMPILEAZA linia 10 da eroare

```
   #include <iostream>
   using namespace std;
   template<class X>
   X f(X x, X y)
   { return *x-*y; }
   int f(double *x, int y)
   { return *(x+y); }
   int main()
   { double a=7.7,*b=new double(21);
     cout<<f(a,a);
     return 0; }

   - fara conversie la non-template 0.1p
   -fara conversie la template 0.1p
   -*x si *y fara sens pt x si y double 0.2p
   -varianta corecta 0.1p
```

Problema 12:

Varianta 1: identic cu varianta 3

```
#include <iostream>
using namespace std;
class A
{ int x;
  public:
    A(int i=2):x(i){}
    int get_x(){ return x;}
    A operator+(int);
};
ostream& operator<<(ostream& o, A a)
   {o<<a.get_x(); return o; }
A A::operator+(int i){return x+i;}
int main()
{ A a=-33; int b=44;
  cout<<a+b<<" "<<b+3;
    return 0;}
```

Varianta 2: NU COMPILEAZA ca in operatorul << avem a.x si nu avem acces la x

```
#include <iostream>
using namespace std;
class A
{ int x;
  public:
    A(int i=2):x(i){}
    int get_x(){ return x;}
    A operator+(int);
};
ostream& operator<<(ostream& o, A a)
   {o<<a.x; return o; }
A A::operator+(int i){return x+i;}
int main()
{ A a=-33; int b=44;
  cout<<a+b<<" "<<b+3;
    return 0;}
-se apeleaza constructorul de initializare cu -33 si
 se apeleaza op + supraincarcat 0.1p
-se intoarce un int care e convertit prin constructor la un obiect de
tip A 0.1p
-pentru acesta se apeleaza op << supraincarcat
- a.x e privat 0.1p
-varianta corecta 0.1p
```

Varianta 3: COMPILEAZA si afiseaza 11 47

```
#include <iostream>
using namespace std;
class A
{ int x;
```

```
  public:
    A(int i=2):x(i){}
    int get_x(){ return x;}
    A operator+(int);
};
ostream& operator<<(ostream& o, A a)
   {o<<a.get_x(); return o; }
A A::operator+(int i){return x+i;}
int main()
{ A a=-33; int b=44;
  cout<<a+b<<" "<<b+3;
    return 0;}
```

-se apeleaza constructorul de initializare cu -33 0.1p
- se apeleaza op + supraincarcat 0.1p
-se intoarce un int care e convertit prin constructor la un obiect de tip A 0.1p
-pentru acesta se apeleaza op << supraincarcat si op << pt int 0.1p
- rezultatul 11=-33+44 47=44+3 0.1 p

Varianta 4: COMPILEAZA si afiseaza -11 -41

```
#include <iostream>
using namespace std;
class A
{ int x;
  public:
    A(int i=2):x(i){}
    int get_x(){ return x;}
    A operator+(int);
};
ostream& operator<<(ostream& o, A a)
   {o<<a.get_x(); return o; }
A A::operator+(int i){return x+i;}
int main()
{ A a=33; int b=-44;
  cout<<a+b<<" "<<b+3;
    return 0;}
```

-se apeleaza constructorul de initializare cu 33 0.1p
- se apeleaza op + supraincarcat 0.1p
-se intoarce un int care e convertit prin constructor la un obiect de tip A 0.1p
-pentru acesta se apeleaza op << supraincarcat si op << pt int 0.1p
-rezultatul -11=33-44 -41=-44+3 0.1p

Varianta 5: COMPILEAZA si afiseaza -77 -47

```
#include <iostream>
using namespace std;
class A
{ int x;
  public:
    A(int i=2):x(i){}
    int get_x(){ return x;}
    A operator+(int);
```

```
};
ostream& operator<<(ostream& o, A a)
   {o<<a.get_x(); return o; }
A A::operator+(int i){return x+i;}
int main()
{ A a=-33; int b=-44;
  cout<<a+b<<" "<<b-3;
    return 0;}
```

-se apeleaza constructorul de initializare cu -33 0.1p
- se apeleaza op + supraincarcat 0.1p
-se intoarce un int care e convertit prin constructor la un obiect de tip A 0.1p
-pentru acesta se apeleaza op << supraincarcat si op << pt int 0.1p
-rezultatul -77=-33-44 -47=-44-3 0.1p

Q13 :
Descrieţi pe scurt caracteristicele fiecăreia și diferenţele dintre transmiterea parametrilor unei funcţii prin pointer, respectiv prin referinţă.

-sintaxa declaratie functie + apel functie pt pointer si referinta 0.2p
-se modifica parametrul actual transmis prin pointer sau referinta 0.1p
-nu se copiaza obiectul(nu se apeleaza constructorul de copiere)0.1p
-permite mecanismul metodelor virtuale 0.1p
-accesare directa(referinta) si prin dereferentiere (pointer)0.1
-definitie pointer si referinta 0.1p
-transmitere prin valoare (pointer) cu copierea pointerului pe stiva si transmitere alt nume (referinta) 0.1

in caz ca nu apare nici una din cele de mai sus:

-referintele nu pot fi nule , pointerii pot fi nuli 0.1
-referintele nu isi schimba zona referita, pointerii si-o pot schimba 0.1

Teorie:

14. **Descrieți pe scurt supraîncărcarea operatorilor unari (sintaxă, exemple, particularități)**.

a. Sintaxa pentru supraîncărcarea operatorilor ca funcții membre și ca funcții prieten 0.1
b. Exemple pentru supraîncărcarea operatorilor ca funcții membre și ca funcții prieten 0.1
c. Supraîncărcarea operatorilor prefixați și postfixați ca funcții membre 0.1
d. Supraîncărcarea operatorilor prefixați și postfixați ca funcții prieten 0.1

Exemplificare de utilizare 0.1

16. Descrieți pe scurt utilizarea funcțiilor inline (recomandări, exemple, particularități).

a. nu sunt apelate efectiv, sunt similare macrodefinitiilor – 0.1
b. pot fi declarate explicit prin cuvântul cheie *inline* , sau implicit când se definesc în clasa – 0.1
c. nu se pot folosi structurile repetitive într-o funcție *inline* +functiile recursive nu pot fi *inline* – 0.1
d. dezavantaj – cod mare prin duplicare, deci se recomanda funcții mici – 0.1
e. exemple inline explicit și implicit – 0.1

Q15 :
Descrieţi pe scurt cum se poate realiza conversia de la un tip de date de bază la tipul definit de o clasă nouă (sintaxă, utilizare, particularităţi, exemplu).
-constructor cu un parametru de tip de baza la clasa noua 0.1 + sintaxa 0.1
-operator cast dintr-o clasa spre clasa noua 0.1+ sintaxa 0.1
-operator = cu un parametru de tipul de baza, in clasa noua 0.1 +sintaxa 0.1
-utilizare -instructiunea care face sa se execute conversia 0.1
-exemplu -care sa completeze campuri in clasa noua 0.1

daca nu apar cele de mai sus: (desi face coversia de pointeri sau referinte!)
-dynamic_cast 0.1 +sintaxa corecta 0.1

17:
Descrieţi pe scurt trei moduri prin care se poate schimba starea obiectului dintr-o funcţie de instanţă constantă.
Pointerul this cu cast away constness 0.1 sau 0.2 (depinde de cat de bine e explicat)
sau const_cast 0.1
O bucata definita mutable 0.1
O bucata definita volatile 0.1
prostii -0.1

18:
Descrieţi din punctul de vedere al programării OOP o clasă care nu poate fi instanţiată decât o singură dată (definiţie, utilizare, proprietăţi, exemplu).

spus (singleton, design pattern creational), 0.1
pentru constructor privat ca sa se controleze instantierea din afara clasei, 0.1
funcţie statică de creare obiect/verificare 0.1
alocare dinamică plus pointer sau referinta statica 0.1
dat exemplu de utilizare: 0.1
prostii -0.1 pe prostie
nedefinit pointerul static in afara clasei -0.1