



1

1 Specificați și testați funcția: (1.5p)

```
int f(int x) {  
    if (x <= 0)  
        throw std::exception("Invalid argument!");  
  
    int rez = 0;  
    while (x)  
    {  
        rez = rez * 10 + x % 10;  
        x /= 10;  
    }  
    return rez;  
}
```

palindrom

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)  
#include <iostream>  
using namespace std;  
int except(bool thrEx) {  
    if (thrEx) {  
        throw 2;  
    }  
    return 3;  
}  
  
int main() {  
    try {  
        cout << except(1 < 1); 3  
        cout << except(true); 2  
        cout << except(false); -  
    } catch (int ex) {  
        cout << ex;  
    }  
    cout << 4; 4  
    return 0;  
}
```

```
//2 b (0.5p)  
#include <iostream>  
using namespace std;  
class A {  
public:  
    A() {cout << "A" << endl;}  
    ~A() {cout << "~A" << endl;}  
    void print() {  
        cout << "print" << endl;  
    }  
};  
  
void f() {  
    A a[2]; A A  
    a[1].print(); print  
}  
int main() {  
    f(); ~A ~A  
    return 0;  
}
```

324

AA print ~A ~A

1. /*

Functia calculeaza rasturnatul unui numar.

Preconditii: $x \leq 0$, altfel arunca exceptie

Param de intrare: x - intreg

Resurse: rez - intreg

Postconditii: rez sa fie inversul lui x

*/

void test()

{

assert(f(12) == 21);

assert(f(123) == 321);

try

{

f(0);

assert(false);

y

catch (exception)

{

assert(true);

z

try

{

f(-2);

assert(false);

y

catch (exception)

{

assert(true);

z

int main()

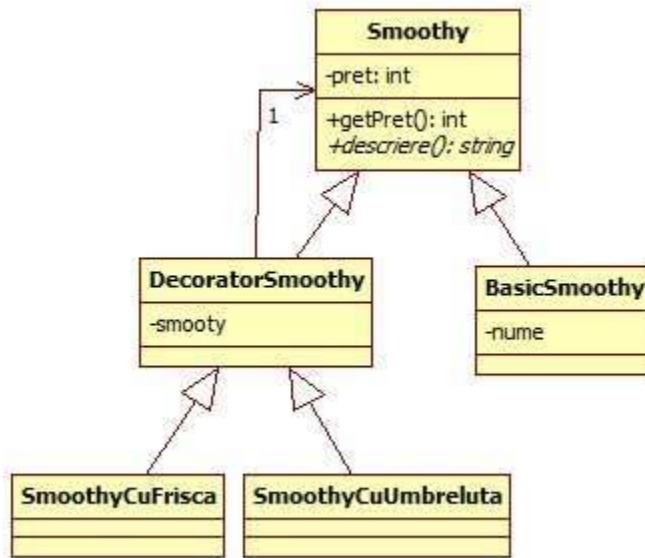
{

test();

return 0;

}

3) Scrieți codul C++ ce corespunde diagramei de clase UML. (4p)



- Clasa abstractă **Smoothy** are o metodă pur virtuală descriere(). **DecoratorSmoothy** conține un smoothy, metodele descriere() și getPret() returnează descrierea și prețul smoothy-ului agregat.
- Clasele **SmoothyCuFrisca** și **SmoothyCuUmbreluta** adaugă textul “cu frisca” respectiv “cu umbrelută” la descrierea smoothy-ului conținut. Prețul unui smoothy care are frisca crește cu 2 Ron, cel cu umbreluță costa în plus 3 RON.
 - Clasa **BasicSmoothy** reprezintă un smoothy fără frișcă și fără umbreluță, metoda descriere() returnează denumirea smoothy-ului.

Se cere:

1 Codul C++ doar pentru clasele: **Smoothy**, **DecoratorSmoothy**, **SmoothyCuFrisca** (0.75) ✓
 2 Scrieți o funcție C++ care returnează o listă de smoothy-uri: un smoothy de kivi cu frișcă și umbrelută, un smoothy de căpsuni cu frișcă și un smoothy simplu de kivi. (0.5p)

3 Programul principal apelează funcția descrisă mai sus, apoi tipărește descrierea și prețul pentru fiecare smoothy în ordine alfabetica după descriere. (0.25p)

• Creați doar metode și atribută care rezultă din diagrama UML (adăugați doar lucruri specifice C++ ex: constructori). Nu adăugați câmpuri, metode, nu schimbați vizibilitatea, nu folosiți friend. Folosiți STL unde există posibilitatea.

• Detalii barem: **1.5p** Polimorfism, **1p** Gestiona memoriai, **1.5p** Restul(defalcat mai sus)

4 Definiți clasa Geanta astfel încât următoarea secvență C++ să fie corecta sintactic și să efectueze ceea ce indica comentariile. (2p)

```

void calatorie() {
    Geanta<string> ganta{ "Ion" }; //creaza geanta pentru Ion
    ganta = ganta + string{ "haine" }; //adauga obiect in ganta
    ganta + string{ "pahar" };
    for (auto o : ganta) { //itereaza obiectele din geanta
        cout << o << "\n";
    }
}
  
```

```
3. class Smoothy
{
private:
    int pret;
public:
    Smoothy (int p) : pret {p} {};
    virtual int getPret () {
    {
        return this->pret;
    }
    virtual string descriere () = 0;
    virtual ~Smoothy () = default;
};

class BasicSmoothy : public Smoothy
{
private:
    string nume;
public:
    BasicSmoothy (int pret, string n) : Smoothy (pret), nume {n} {};
    string descriere () override {
    {
        return nume;
    }
};

class DecoratorSmoothy : public Smoothy
{
private:
    Smoothy *s;
public:
    DecoratorSmoothy (Smoothy *s0) : Smoothy (s0->getPret()), s {s0} {};
    virtual string descriere () override {
    {
        return s->descriere ();
    }
    virtual int getPret () override {
    {
        return s->getPret ();
    }
    virtual ~DecoratorSmoothy () {
    {
        delete s;
    }
};
```

```

class SmoothyCuFriza : public DecoratorSmoothy
{
public:
    SmoothyCuFriza (Smoothy *s) : DecoratorSmoothy (s) {}
    string descriere() override
    {
        return DecoratorSmoothy::descriere () + " cu friza";
    }
    int getRet() override
    {
        return DecoratorSmoothy::getRet() + 2;
    }
};

```

```

class SmoothyCuUmbreluta : public DecoratorSmoothy
{
public:
    SmoothyCuUmbreluta (Smoothy *s) : DecoratorSmoothy (s) {}
    string descriere() override
    {
        return DecoratorSmoothy::descriere () + " cu umbreluta";
    }
    int getRet() override
    {
        return DecoratorSmoothy::getRet() + 3;
    }
};

```

3.2. vector < Smoothy* > functie2()

```

{
    vector < Smoothy* > rez;
    rez.push_back (new SmoothyCuUmbreluta (new SmoothyCuFriza (new BasicSmoothy {1, "kiu!"})));
    rez.push_back (new SmoothyCuFriza (new BasicSmoothy {2, "capsu!"}));
    rez.push_back (new BasicSmoothy {3, "kiki!"});
    return rez;
}

```

3.3. int main()

```

{
    vector < Smoothy* > rez = functie2();
    for (int i=0, i<rez.size()-1, i++)
        for (int j=i+1, j<rez.size(); j++)
            if (rez[i] -> descriere() > rez[j] -> descriere ())
                swap (rez[i], rez[j]);
    for (auto s : rez)
    {
        cout << s -> descriere() << " " << s -> getRet() << "\n";
        delete s;
    }
    ~CrtDumpMemoryLeaks();
    return 0;
}

```

4 Definiți clasa Geanta astfel încât următoarea secvență C++ sa fie corecta sintactic și sa efectueze ceea ce indica comentariile. (2p)

```
void calatorie() {
    Geanta<string> ganta{ "Ion" }; //creaza geanta pentru Ion
    ganta = ganta + string{ "haine" }; //adauga obiect in ganta
    ganta + string{ "pahar" };
    for (auto o : ganta) { //itereaza obiectele din geanta
        cout << o << "\n";
    }
}
```

4. class Geanta

```
{ private:
    string owner;
    vector<TElem> obiecte;
public:
    Geanta(string nume): owner{nume} {};
    Geanta & operator+(const TElem & obj)
    {
        obiecte.push_back(obj);
        return *this;
    }
    typename vector<TElem>::iterator begin()
    {
        return obiecte.begin();
    }
    typename vector<TElem>::iterator end()
    {
        return obiecte.end();
    }
};
```

1 Specificați și testați funcția: (1.5p)

```
std::pair<int, int> f(std::vector<int> l) {
    if (l.size() < 2) throw std::exception{};
    std::pair<int, int> rez{-1, -1};
    for (auto el:l) {
        if (el < rez.second) continue;
        if (rez.first < el) {
            rez.second = rez.first;
            rez.first = el;
        }else{
            rez.second=el;
        }
    }
    return rez;
}
```

cele mai mari două numere
dim vector
descrescător

1, 3, 2, 1

rez = -1, -1

1 -1

3 1

3 2

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)
#include <iostream>
#include <vector>
struct A {
    A() {std::cout << "A";}
    virtual void print() {
        std::cout << "A";
    }
};
struct B : public A{
    B() { std::cout << "B"; }
    void print() override {
        std::cout << "B";
    }
};
int main() {
    std::vector<A> v; A
    v.push_back(A{}); A
    v.push_back(B{}); B
    for (auto& el : v) el.print(); AA
    return 0;
}
```

```
//2 b (0.5p)
#include <iostream>
using namespace std;
class A {
    int x;
public:
    A(int x) : x{ x } {}
    void print(){cout<< x << endl;}
};
A f(A a) {
    a.print(); 4
    a = A{ 10 };
    a.print(); NO
    return a;
}
int main() {
    A a{ 4 };
    a.print(); 4
    f(a);
    a.print(); 4
}
```

AABAA

4 4 10 4

1. /*

Functia returneaza cele mai mari doua elemente din vector.

Date de intrare: l - vector de intregi

Date de ieșire: o perche de intregi, cele mai mari două numere

dacă vectorul are lungimea < 2 va arunca o excepție

ace două el negative va returna -1,-1

*/

void test()

{

std::vector<int> li;

try

{

f(li);

assert(false);

}

catch (std::exception)

{

assert(true);

}

std::vector<int> l {1,3,5,7,9};

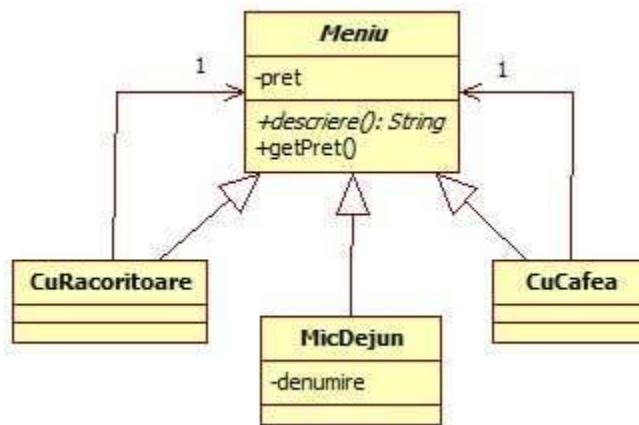
std::pair<int,int> p = f(l);

assert(p.first == 9);

assert(p.second == 7);

}

3 Scrieți codul C++ ce corespunde diagramei de clase UML. (4p)



- Clasa abstractă **Meniu** are o metodă pur virtuală descriere()
- **CuRacoritoare** și **CuCafea** conțin un meniu și metoda descriere() adaugă textul “cu racoritoare” respectiv “cu cafea” la descrierea meniului conținut. Prețul unui meniu care conține răcoritoare crește cu 4 Ron, cel cu cafea costa în plus 5 RON.
- Clasa **MicDejun** reprezintă un meniu fără răcoritoare și fără cafea, metoda descriere() returnează denumirea meniului. În restaurant pizzerie există 2 feluri de mic dejun: Ochiuri și Omleta, la prețul de 10 respectiv 15 RON.

Se cere:

- 1 Codul C++ doar pentru clasele: **Meniu**, **CuCafea** (0.75)
 - 2 Scrieți o funcție C++ care returnează o listă de meniuri: un meniu cu răcoritoare și cafea, un meniu cu Ochiuri și cafea, un meniu cu Omleta. (0.5p)
 - 3 În programul principal se creează o comandă (folosind funcția descrisă mai sus), apoi se tipărește descrierea și prețul pentru fiecare pizza în ordinea descrescătoare a prețurilor. (0.25p)
- Creați doar metode și attribute care rezultă din diagrama UML (adăugați doar lucruri specifice C++ ex: constructori). Nu adăugați câmpuri, metode, nu schimbați vizibilitatea, nu folosiți friend. Folosiți STL unde există posibilitatea.

Detalii barem: **1.5p** Polimorfism, **1p** Gestiona memoria, **1.5p** Restul(defalcat mai sus)

4 Definiți clasa Measurement astfel încât următoarea secvență C++ să fie corectă sintactic și să efectueze ceea ce indica comentariile. (2p)

```

int main() {
    //creaza un vector de masuratori cu valorile {10,2,3}
    std::vector<Measurement<int>> v{ 10,2,3 };
    v[2] += 3 + 2; //aduna la masuratoarea 3 valoarea 5
    std::sort(v.begin(), v.end()); //sorteaza masuratorile
    //tipareste masuratorile (in acest caz: 2,8,10)
    for (const auto& m : v) std::cout << m.value() << ",";
    return 0;
}
  
```

3. class Meniu

```
{  
    private:  
        int pret;  
    public:  
        Meniu (int pret) : pret (pret) {}  
        virtual string descriere () = 0;  
        virtual int getPret ()  
        {  
            return pret;  
        }  
        virtual ~Meniu () = default;  
};
```

class CuCafea : public Meniu

```
{  
    private:  
        Meniu *menu;  
    public:  
        CuCafea (Meniu *m) : Meniu (m->getPret ()) , menu (m) {}  
        string descriere () override  
        {  
            return menu->descriere () + " cu cafea";  
        }  
        int getPret () override  
        {  
            return menu->getPret () + 5;  
        }  
        ~CuCafea () { delete menu; }  
};
```

class MicDejum : public Meniu

```
{  
    private:  
        string nume;  
    public:  
        MicDejum (string nume, int pret) : Meniu (pret) , nume (nume) {}  
        string descriere () override  
        {  
            return nume;  
        }  
};
```

3.2. vector < Meniu * > functie2()

{
vector < Meniu * > rez;

rez.push_back (new Culafca (new CuRacintuire (new MicDejum ("Omleta", 15))));

rez.push_back (new Culafca (new MicDejum ("Ochiwi", 10)));

rez.push_back (new MicDejum ("Omleta", 15));

return rez;

}

3.3. int main ()

{

vector < Meniu * > rezultat = functie2();

sort (rezultat.begin(), rezultat.end(), [] (Meniu * o1, Meniu * o2)

{

return o1->getPret() > o2->getPret();

});

for (auto m : rezultat)

{

cout << m->descriere() << " " << m->getPret() << "\n";

delete m;

}

return 0;

}

4 Definiți clasa Measurement astfel încât următoarea secvență C++ sa fie corecta sintactic si sa efectueze ceea ce indica comentariile. (2p)

```
int main() {
    //creaza un vector de masuratori cu valorile {10,2,3}
    std::vector<Measurement<int>> v{ 10,2,3 };
    v[2] + 3 + 2; //aduna la masuratoarea 3 valoarea 5
    std::sort(v.begin(), v.end()); //sorteaza masuratorile
    //tipareste masuratorile (in acest caz: 2,8,10)
    for (const auto& m : v) std::cout << m.value() << ",";
    return 0;
}
```

4. class Measurement

{

private:

TElem val;

public:

Measurement(TElem v): val{v} {};

TElem value() const

{

return val;

}

Measurement <TElem> & operator+(const Measurement <TElem> & e)

{

val = val + e.val;

return *this;

}

friend bool operator<(const Measurement <TElem> & e1, const Measurement <TElem> & e2)

{

return e1.val < e2.val;

}

y;

(3)

1 Specificați și testați funcția: (1.5p)

```
bool f(int a) {
    if (a <= 0)
        throw std::exception("Illegal argument");
    int d = 2;
    while (d<a && a % d>0) d++;
    return d>=a;  ↗(true)
}
```

$a = \text{prim} \Rightarrow 1$

$a = \text{par} \Rightarrow 0$
 $a \leq 2 \Rightarrow 1$

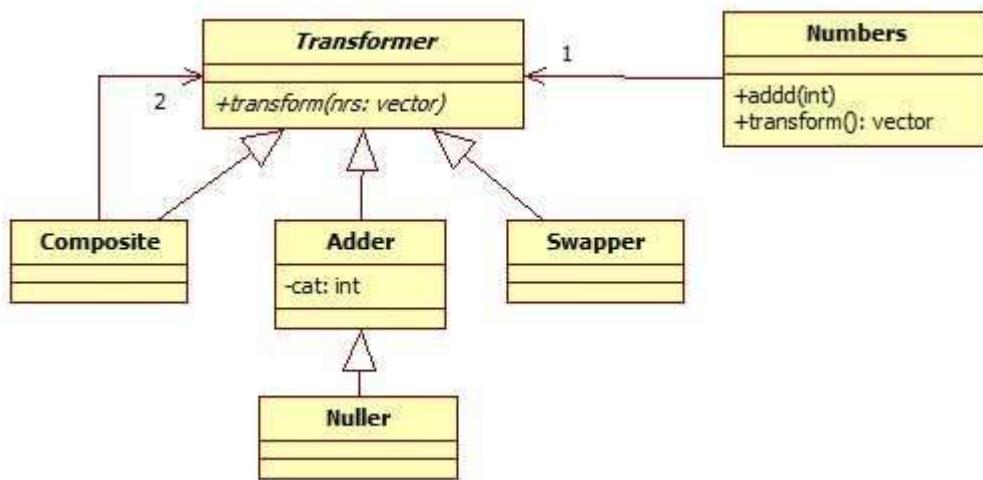
2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)
#include <iostream>
using namespace std;
class A {
public:
    A(){cout << "A()" << endl;}
    void print() {cout << "printA" << endl;}
};
class B: public A {
public:
    B(){cout << "B()" << endl;}
    void print() {cout << "printB" << endl;}
};
int main() {
    A* o1 = new A(); A()
    B* o2 = new B(); A() B()
    o1->print(); printA
    o2->print(); printB
    delete o1; delete o2;
    return 0;
}
```

```
//2 b (0.5p)
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> v;
    v.push_back(5);
    v.push_back(7);
    v[0] = 6;
    v.push_back(8);
    auto it = v.begin();
    it = it + 1;
    while (it != v.end()) {
        cout << *it << endl;
        it++;
    }
    return 0;
}
```

A() A() B() printA printB 7,8

3 Scrieți codul C++ ce corespunde diagramei de clase UML. (4p)



- Clasa abstractă **Transformer** are o metoda pur virtuala `transform(nrs)`
- Metoda `transform()` din clasa **Adder** adaugă la fiecare număr un număr dat (`cat`) , metoda `transform` din **Swapper** interschimbă numere consecutive (poziția 0 cu poziția 1, poziția 2 cu 3, etc) iar `transform()` din clasa **Nller** înlocuiește numărul cu 0 dacă în urma aplicării adunării numărul este > 10 sau lasă numărul ce rezultă în urma adunării. Clasa **Composite** în metoda `transform()` aplică succesiv cele două transformări folosind **Transformer**-ele agregate.
- Metoda `transform()` din clasa **Numbers** ordonează descrescător numerele adăugate cu `add` și apelează metoda `transform(nrs)` din **Transformer**-ul continut.

Se cere:

- 1 Codul C++ doar pentru clasele: **Transformer**, **Composite**, **Nller** (0.75p)
 - 2 Scrieți o funcție fiecare creează și returnează un obiect **Numbers** care compune un **Nller** (`cat=9`) cu un **Swapper** compus cu un **Adder** (`cat=3`). (0.5p)
 - 3 În funcția main apelați funcțiile de mai sus, adăugați cale 5 numere în cele două obiecte **Numbers**. Apoi apelați funcția `transform` pentru ambele. (0.25p)
- Creați doar metode și attribute care rezulta din diagrama UML (adăugați doar lucruri specifice C++ ex: constructori). Nu adăugați câmpuri, metode, nu schimbați vizibilitatea, nu folosiți friend. **Barem: 1.5p Polimorfism, 1p Gestionearea memoriei, 1.5p Defalcat mai sus**

4 Definiți clasele **ToDo** și **Examen** general astfel încât următoarea secvență C++ să fie corectă sintactic și să efectueze ceea ce indică comentariile. (2p)

```

void todolist() {
    ToDo<Examen> todo;
    Examen oop{ "oop scris", "8:00" };
    todo << oop << Examen{"oop practic", "11:00"};      //Adauga 2 examene la todo
    std::cout << oop.getDescriere(); //tipareste la consola: oop scris ora 8:00
    //iterarea elementelor adăugate și tipareste la consola lista de activități
    //în acest caz tipareste: De facut:oop scris ora 8:00;oop practic ora 11:00
    todo.printToDoList(std::cout);
}
  
```

3. class Transformer

```
{  
    public:  
        virtual void transform (vector <int> & mrs) = 0;  
        virtual ~Transformer () = default;  
};
```

class Adder : public Transformer

```
{  
    private:  
        int cat;  
    public:  
        Adder (int c) : cat {c} {};  
        virtual void transform (vector <int> & mrs)  
        {  
            for (int i=0; i< mrs.size(); i++)  
                mrs[i] += thus > cat;  
        }  
};
```

class Swapper : public Transformer

```
{  
    public:  
        void transform (vector <int> & mrs)  
        {  
            for (int i=0; i<= mrs.size() - 2; i+=2)  
            {  
                int aux = mrs[i];  
                mrs[i] = mrs[i+1],  
                mrs[i+1] = aux;  
            }  
        }  
};
```

class Nuller : public Adder

```
{  
    public:  
        Nuller (int cat) : Adder {cat} {};  
        void transform (vector <int> & mrs)  
        {  
            Adder::transform (mrs);  
            for (int i=0; i< mrs.size(); i++)  
                if (mrs.at(i) > 10)  
                    mrs[i] = 0;  
        }  
};
```

```
class Composite : public Transformer
{
private:
    Transformer *t1,
    Transformer *t2,
public:
    Composite (Transformer *t1, Transformer *t2) : t1(t1), t2(t2) {};
    void transform (vector <int> &res)
    {
        t1->transform (res),
        t2->transform (res);
    }
    ~Composite ()
    {
        delete t1;
        delete t2;
    }
};
```

```
class Numbers
{
private:
    Transformer *t,
    vector <int> v,
public:
    Numbers (Transformer *t0) : t(t0), v();
    Numbers (Numbers &t) : t(t), v();
    void add (int elem)
    {
        v.push_back (elem);
    }
    vector <int> &transform ()
    {
        sort (v.begin(), v.end(), [] (int a, int b)
        {
            return a > b;
        });
        this->t->transform (v);
        return v;
    }
    ~Numbers ()
    {
        if (t != NULL)
            delete t;
    }
};
```

3.2. Numbers functie2()

{

 Numbers m { new Composite { new Nuller {} }, new Composite { new Swapper, new Adder {} } };
 return m;

}

3.3. int main()

{

 Numbers m = functie2();
 m.addd(1);
 m.addd(1);
 m.addd(1);
 m.addd(1);
 m.addd(1);
 vector<int> v = m.toString();
 for (auto nr : v)
 cout << nr << " ";
 _CrtDumpMemoryLeaks();

 return 0;

}

4 Definiți clasele ToDo si Examen general astfel încât următoarea secvență C++ sa fie corecta sintactic si sa efectueze ceea ce indică comentariile. (2p)

```
void todolist() {
    ToDo<Examen> todo;
    Examen oop{ "oop scris", "8:00" };
    todo << oop << Examen{"oop practic", "11:00"};      //Adauga 2 examene la todo
    std::cout << oop.getDescriere(); //tipareste la consola: oop scris ora 8:00
    //iterarea elementele adaugate si tipareste la consola lista de activitati
    //in acest caz tipareste: De facut:oop scris ora 8:00;oop practic ora 11:00
    todo.printToDoList(std::cout);
}
```

```
class Examen
{
private:
    string nume,
public:
    Examen(const string& nume1, const string& ora)
    {
        this->nume = nume1 + " " + ora,
    }
    const string& getDescriere() const
    {
        return this->nume;
    }
};

class ToDo
{
private:
    vector<TElem> lista;
public:
    ToDo operator<< (const TElem& elem)
    {
        lista.push_back(elem);
        return *this;
    }
    void printToDoList (ostream& out)
    {
        out << "De făcut: ";
        int nr = lista.size();
        for (int i=0; i<nr; i++)
        {
            out << lista[i].getDescriere();
            if (i!=nr-1)
                out << ",";
        }
    }
};
```

(4)

1 Specificați și testați funcția: (1.5p)

```
bool f(int a) {  
    if (a <= 1) m prim sau nu  
        throw "Illegal argument";  
    int aux = 0;  
    for (int i = 2; i < a; i++) {  
        if (a % i == 0) {  
            aux++;  
        }  
    }  
    return aux == 0;  
}
```

$a \leq 1 \Rightarrow \text{exc}$
 $a = \text{prim} \Rightarrow 1$

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

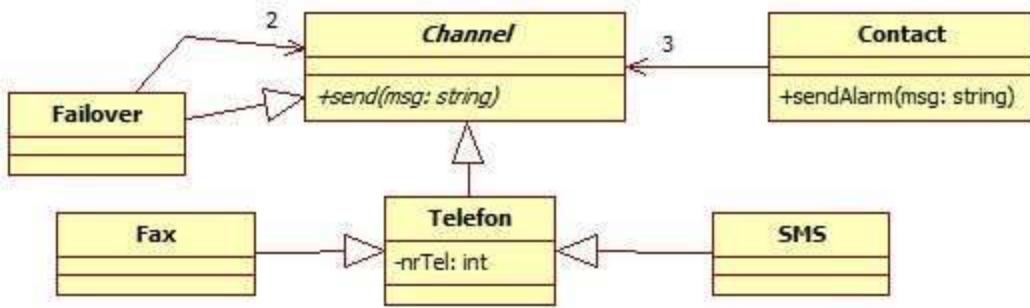
```
//2 a (1p)  
#include <vector>  
#include <iostream>  
using namespace std;  
class A {  
public:  
    virtual void f() = 0;  
};  
class B:public A{  
public:  
    void f() override {  
        cout << "f din B";  
    }  
};  
class C :public B {  
public:  
    void f() override {  
        cout << "f din C";  
    }  
};  
int main() {  
    vector<A> v; *  
    B b;  
    v.push_back(b);  
    C c;  
    v.push_back(c);  
    for (auto e : v) { e.f(); }  
    return 0;  
}
```

```
//2 b (0.5p)  
#include <iostream>  
using namespace std;  
class A {  
public:  
    A() {cout << "A" << endl;}  
    ~A() {cout << "~A" << endl;}  
    void print() {cout << "print" <<  
    endl;}  
};  
void f() {  
    A a[2]; AA  
    a[0].print(); print  
}  
int main() {  
    f(); ~A ~A  
    return 0;  
}
```

eroare \rightarrow A este clasa abstractă

AA print ~A ~A

3 Scrieți codul C++ ce corespunde diagramei de clase UML. (4p)



- Clasa abstractă **Channel** are o metoda pur virtuala *send*
 - Metoda *send* din clasa **Telefon** tipărește mesajul “dail.” si numărul de telefon conținut, dar din când în când (în funcție de un număr aleator generat) aruncă excepție std::exception indicând ca linia este ocupată.
 - Clasa **Fax** și **SMS** încearcă să apeleze numărul de telefon și în caz de succes tipărește “sending fax” respectiv “sending sms”. Clasa **Failover** încearcă să trimită mesajul pe primul canal, dacă trimiterea eșuează (este ocupat) atunci încearcă trimitera pe canalul secundar.
 - Metoda *sendAlarm* din clasa **Contact**, încearcă să trimită repetat mesajul pe cele 3 canale conținute pe rând până reușește trimitera (găsește o linie care nu este ocupată).
- Se cere:

1 Codul C++ doar pentru clasele: Channel, Failover, Fax, Contact(0.75p)

2 Scrieți o funcție C++ care creează și returnează un obiect **Contact cu următoarele canale (alegeți voi numere de telefon): 1 Telefon; 2 Fax “daca este ocupat încearcă” Sms ; 3 Telefon “daca este ocupat încearcă” Fax “daca este ocupat încearcă” SMS. (0.5p)**

3 În funcția main apelați funcția de mai sus și trimiteți 3 mesaje. (0.25p)

- Creati doar metode si atribute care rezulta din diagrama UML (adăugați doar lucruri specifice C++ ex: constructori). Nu adăugați câmpuri, metode, nu schimbați vizibilitatea, nu folosiți friend. Folosiți STL unde există posibilitatea.

Detalii barem: **1.5p** Polimorfism, **1p** Gestiuinea memoriei, **1.5p** Restul(Defalcat mai sus)

4 Definiți clasa Expresie generală astfel încât următoarea secvență C++ sa fie corecta sintactic și să efectueze ceea ce indică comentariile. (2p)

```

void operatii() {
    Expresie<int> exp{ 3 }; //construim o expresie,pornim cu operandul 3
    //se extinde expresia in dreapta cu operator (+ sau -) si operand
    exp = exp + 7 + 3;
    exp = exp - 8;
    //tipareste valoarea expresiei (in acest caz:5 rezultat din 3+7+3-8)
    cout << exp.valoare() << "\n";
    exp.undo(); //reface ultima operatie efectuata
    //tipareste valoarea expresiei (in acest caz:13 rezultat din 3+7+3)
    cout << exp.valoare() << "\n";
    exp.undo().undo();
    cout << exp.valoare() << "\n"; //tipareste valoarea expresiei (in acest caz:3)
}

```

```
3. class Channel
```

```
{  
    public:  
        virtual void send(string msg) = 0;  
        virtual ~Channel() = default;  
};
```

```
class Telefon: public Channel
```

```
{  
    private:  
        int nrTel;  
    public:  
        Telefon(int nr): nrTel(nr){};  
        virtual void send(string msg) override  
        {  
            int nr = rand();  
            if(nr == 0)  
                throw exception("Linia este ocupata!\n");  
            cout << "dial " << nrTel << "\n";  
        };  
};
```

```
class Fax: public Telefon
```

```
{  
    public:  
        Fax(int nr): Telefon(nr){};  
        void send(string msg) override  
        {  
            try  
            {  
                Telefon::send(msg);  
                cout << " sending fax";  
            }  
            catch(exception)  
            {  
                throw exception("Linia este ocupata!\n");  
            };  
        };  
};
```

SMS la fel

```
class Failover : public Channel
```

```
{ private:
```

```
    Channel * c1,  
    Channel * c2;
```

```
public:
```

```
    Failover (Channel * c01, Channel * c02) : m1(c01), m2(c02) {};
```

```
    Failover (Failover && ot) noexcept
```

```
{
```

```
    c1 = ot.c1;  
    c2 = ot.c2;  
    ot.c1 = nullptr;  
    ot.c2 = nullptr;
```

```
} void send (string msg) override
```

```
{
```

```
    try
```

```
{
```

```
    c1->send (msg);
```

```
}
```

```
catch (exception)
```

```
{
```

```
    try
```

```
{
```

```
    c2->send (msg);
```

```
}
```

```
catch (exception)
```

```
{
```

```
    cout << "esuat";
```

```
}
```

```
}
```

```
~Failover ()
```

```
{
```

```
    if (c1 != nullptr)
```

```
        delete c1;
```

```
    if (c2 != nullptr)
```

```
        delete c2;
```

```
}
```

```
};
```

```

class Contact
{
private:
    Channel * c1;
    Channel * c2;
    Channel * c3;
public:
    Contact (Channel * c1, Channel * c2, Channel * c3) : c1(c1), c2(c2), c3(c3) {};
    Contact (Contact & ot) : mExcept {
        {
            this->c1 = ot.c1;
            this->c2 = ot.c2;
            this->c3 = ot.c3;
            ot.c1 = nullptr;
            ot.c2 = nullptr;
            ot.c3 = nullptr;
        }
        void sendAlarm (string msg)
        {
            int k=0;
            while (true)
            {
                try
                {
                    if (k==0) c1->send (msg);
                    if (k==1) c2->send (msg);
                    if (k==2) c3->send (msg);
                    break;
                }
                catch (exception)
                {
                    k=(k+1)%3;
                }
            }
        }
        ~Contact ()
        {
            if (c1!=nullptr)
                delete c1;
            if (c2!=nullptr)
                delete c2;
            if (c3!=nullptr)
                delete c3;
        }
    };
}

```

3.2 Contact funtie2()

```
    {  
        Contact c
```

```
        new Telefoni h111y,
```

```
        new Failover (new Fax h222y, new SMS h333y),
```

```
        new Failover (new Telefoni h444y, new Failover h new Fax h555y, new SMS h666y)
```

```
    };
```

```
    return c;
```

```
}
```

3.3 int main()

```
{
```

```
Contact c = funtie2();
```

```
c.send Alarm ("Salut 1");
```

```
cout << '\m';
```

```
c.send Alarm ("Salut 2");
```

```
cout << '\m';
```

```
c.send Alarm ("Salut 3");
```

```
cout << '\m';
```

```
return 0;
```

```
}
```

4 Definiți clasa Expresie generală astfel încât următoarea secvență C++ sa fie corecta sintactic și să efectueze ceea ce indică comentariile. (2p)

```
void operatii()
{
    Expresie<int> exp{ 3 }; //construim o expresie, pornim cu operandul 3
    //se extinde expresia in dreapta cu operator (+ sau -) si operand
    exp = exp + 7 + 3;
    exp = exp - 8;
    //tipareste valoarea expresiei (in acest caz:5 rezultat din 3+7+3-8)
    cout << exp.valoare() << "\n";
    exp.undo(); //reface ultima operatie efectuata
    //tipareste valoarea expresiei (in acest caz:13 rezultat din 3+7+3)
    cout << exp.valoare() << "\n";
    exp.undo().undo();
    cout << exp.valoare() << "\n"; //tipareste valoarea expresiei (in acest caz:3)
}
```

```
class Expresie
{
private:
    vector<pair<TElem, int>> operatii;
public:
    Expresie(TElem elem)
    {
        operatii.push_back({makepair(elem, 1)});
    }
    Expresie & operator+ (const TElem & elem)
    {
        operatii.push_back({makepair(elem, 1)}),
        return *this;
    }
    Expresie & operator- (const TElem & elem)
    {
        operatii.push_back({makepair(elem, -1)}),
        return *this;
    }
    TElem valoare()
    {
        TElem rez = 0;
        for (const auto & el : operatii)
            rez += el.first * el.second;
        return rez;
    }
    Expresie & undo()
    {
        if (operatii.size() != 0)
            operatii.pop_back();
        return *this;
    }
}
```

5

?

1 Specificați și testați funcția: (1.5p)

```
using namespace std;
#include <vector>
#include <string>
#include <algorithm>
#include <map>
vector<int> f(vector<int> l) {
    if (l.size() == 0)
        throw exception("Illegal argument");
    map<int, int> c;
    for (auto e : l) {
        c[e]++;
    }
    sort(l.begin(), l.end(), [&](int a, int b) {
        return c[a] > c[b];
    });
    return l;
}
```

l = 1342

l vidă => exc

2 Indicați rezultatul execuției pentru următoarele programe C++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)
#include <vector>
#include <iostream>
class A {
public:
    A() {
        std::cout << "A";
    }
    virtual void print() {
        std::cout << "printA";
    }
};
class B : public A {
public:
    B() {
        std::cout << "B";
    }
    virtual void print() {
        std::cout << "printB";
    }
};
int main() {
    std::vector<A> v;
    A a; A
    B b; B
    v.push_back(a);
    v.push_back(b);
    for (auto e : v) {e.print();}
    return 0;
}
```

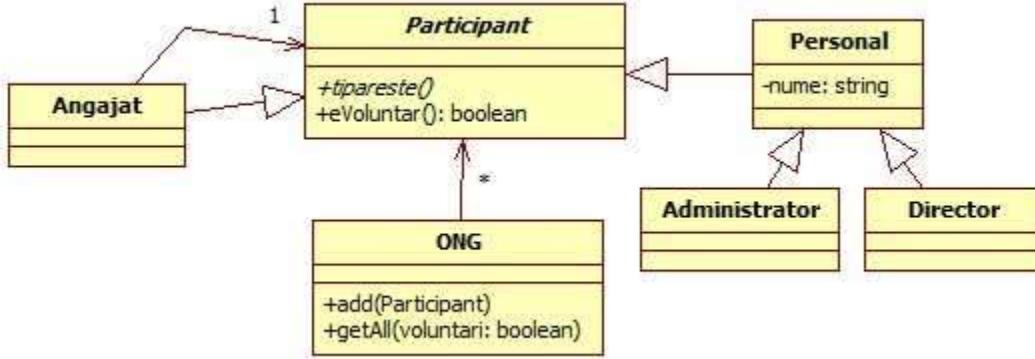
```
//2 b (0.5p)
#include <iostream>
using namespace std;
class A {
public:
    A() {cout << "A" << endl;}
    ~A() {cout << "~A" << endl;}
    void print() {
        cout << "print" << endl;
    }
};
void f() {
    A a[2]; AA
    a[1].print(); print
}
int main() {
    f(); NA NA
    return 0;
}
```

AA print ~A ~A

3 Scrieți codul C++ ce corespunde diagramei de clase UML. (4p)

AAB printA printA

```
1 /* Funcția returnează o copie a vectorului original sortat crescător după frecvența elementelor
 */
2
3 void test () {
4     vector <int> l;
5     try {
6         f(l);
7         assert (false);
8     }
9     catch (exception) {}
10    l.push_back(2); ... 3 5 2 2 3
11    vector <int> rez = f(l);
12    assert (rez.size() == 6); ... 2 2 3 3 5
13    assert (rez[0] == 2);
14 }
```



- Clasa abstractă **Participant** are o metoda pur virtuală *tipareste*.
- Metoda *tipareste* din clasa **Personal** tipărește numele persoanei.
- Clasa **Administrator** și **Director** pe lângă ce tipărește clasa de baza mai tipărește și cuvântul “Administrator” respectiv “Director” .
- Clasa **Angajat** tipărește, pe lângă ce tipărește personalul agregat de el, și textul “angajat”. Metoda *eVoluntar* returnează false.
- Metoda *add* din clasa **ONG** permite adăugarea de orice participant, iar metoda *getAll* returnează doar participanții angajați sau participanții voluntari (în funcție de parametru). Implicit toți participanții sunt voluntari dacă nu sunt decorate cu **Angajat**.

Se cere:

- 1 Codul C++ doar pentru clasele: **Participant**, **Angajat**, **Director**, **ONG**(0.75p)
- 2 O funcție C++ care creează și returnează un obiect **ONG** și adaugă următorii participanți (alegeți voi numele pentru fiecare): un administrator voluntar, un administrator angajat, un director voluntar și un director angajat. (0.5p)
- 3 În funcția main tipăriți separat angajații și voluntarii din ONG. (0.25p)

Creați doar metode și attribute care rezulta din diagrama UML (adăugați doar lucruri specifice C++ ex: constructori). Nu adăugați câmpuri, metode, nu schimbați vizibilitatea, nu folosiți friend. Folosiți STL unde există posibilitatea.

Detalii barem: **1.5p** Polimorfism, **1p** Gestiona memoria, **1.5p** Restul(Defalcat mai sus)

- 4 Definiți clasa Cos generală astfel încât următoarea secvență C++ să fie corectă sintactic și să efectueze ceea ce indică comentariile. (2p)

```

void cumparaturi() {
    Cos<string> cos; //creaza un cos de cumparaturi
    cos = cos + "Mere"; //adauga Mere in cos
    cos.undo(); //elimina Mere din cos
    cos + "Mere"; //adauga Mere in cos
    cos = cos + "Paine" + "Lapte"; //adauga Paine si Lapte in cos
    cos.undo().undo(); //elimina ultimele doua produse adaugate

    cos.tipareste(cout); //tipareste elementele din cos (Mere)
}

```

3. Class Participant

{ protected:

 bool voluntar = true;

public:

 virtual void tipareste() = 0;

 bool eVoluntar()

{

 return voluntar;

}

 virtual ~Participant() = default;

y;

class Personal public Participant

{

private:

 string nume;

public:

 Personal(string n) : nume(n) {};

 void tipareste override

{

 cout << nume << " ";

}

 virtual ~Personal() = default;

y;

class Director : public Personal

{

public:

 Director(string n) : Personal(n) {};

 void tipareste override

{

 Personal::tipareste;

 cout << "Director" << " ";

}

 ~Director() = default;

y;

class Angajat : public Participant

{

private:

 Participant *p;

public:

 Angajat(Participant * pp) : p(pp)

{

 voluntar = false;

}

 void tipareste override

{

 p->tipareste;

 cout << "angajat" << " ";

}

 ~Angajat() { delete p; }

class ONG

{ private:

 vector<Participant*> all;

public:

 ONG() = default;

 void add(Participant * p)

 { all.push_back(p); }

 vector<Participant*> get_all(bool voluntari)

y;

 vector<Participant*> rez;

 for (const auto el : all)

 { if (el->voluntar() == voluntari)

 rez.push_back(el);

 }

 return rez;

y;

3.

3.2. ONG functie2()

y;

ONG o;

o.add(new Administrator {"George"});

o.add(new Angajat(new Administrator {"Vlad"}));

o.add(new Director {"Vasile"});

o.add(new Angajat(new Director {"Horia"}));

return o;

y;

3.3. int main()

y;

ONG o = functie2();

vector<Participant*> voluntari = o.getAll(true);

vector<Participant*> angajati = o.getAll(false);

for (auto el : voluntari)

y;

 el->tipareste();

 cout << endl;

 delete el;

y;

for (auto el : angajati)

y;

 el->tipareste();

 cout << endl;

 delete el;

y;

 return 0;

y;

4 Definiți clasa Cos generală astfel încât următoarea secvență C++ sa fie corecta sintactic si sa efectueze ceea ce indica comentariile. (2p)

```
void cumparaturi() {
    Cos<string> cos;//creaza un cos de cumparaturi
    cos = cos + "Mere"; //adauga Mere in cos
    cos.undo();//elimina Mere din cos
    cos + "Mere"; //adauga Mere in cos
    cos = cos + "Paine" + "Lapte";//adauga Paine si Lapte in cos
    cos.undo().undo();//elimina ultimele doua produse adaugate

    cos.tipareste(cout);//tipareste elementele din cos (Mere)
}
```

```
class Cos
{
private:
    vector<TElem> v;
public:
    Cos()=default;
    Cos & operator + (const TElem & e)
    {
        v.push_back(e);
        return *this;
    }
    Cos & undo()
    {
        v.pop_back();
        return *this;
    }
    void tipareste (ostream & out)
    {
        for (const auto & el: v)
            out << el << " ";
        out << endl;
    }
};
```

(6)

1 Specificați și testați funcția: (1.5p)

rez - lista de divizori $a < 0 \Rightarrow$ exc

```
vector<int> f(int a) {
    if (a < 0)
        throw std::exception("Illegal argument");
    vector<int> rez;
    for (int i = 1; i <= a; i++) {
        if (a % i == 0)
            rez.push_back(i);
    }
    return rez;
}
```

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

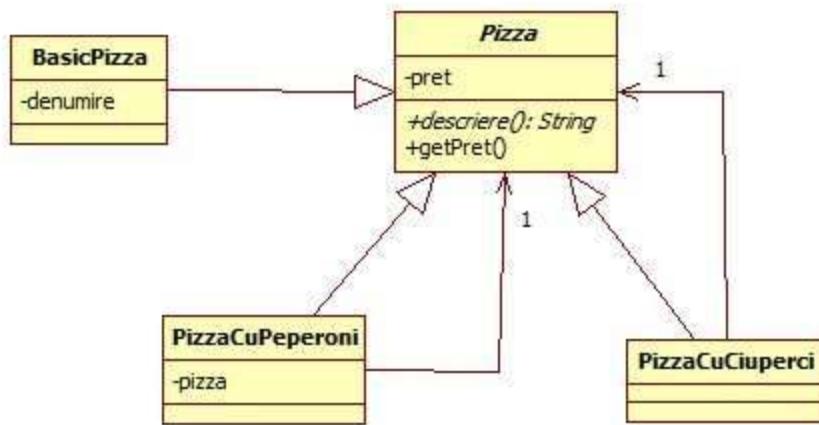
```
//2 a (1p)
#include <iostream>
using namespace std;
int except(int v) {
    if (v < 0) {throw 1; }
    else if (v > 0){
        throw std::exception ("A");
    }
    return 0;
}
int main(){
    try {
        cout << except(1 < 1); ①
        cout << except(-5);
        cout << except(5);-
    }catch (std::exception& e) {
        cout << "A";
    }catch (int x) {
        cout << "B"; ②
    }
    cout << "C"; ③
    return 0;
}
```

```
//2 b (0.5p)
#include <iostream>
using namespace std;
class A {
    int x;
public:
    A(int x) : x{ x } {}
    void print(){cout<< x << ",";};
};
A f(A a) {
    a.print();
    a = A{ 7 };
    a.print();
    return a;
}
int main() {
    A a{ 5 };
    a.print();
    f(a);
    a.print();
}
```

ABC

5,5,7,5,

3 Se da diagrama de clase UML: (4p)



- Clasa abstractă **Pizza** are o metoda pur virtuala descriere()
- **PizzaCuPeperoni** și **PizzaCuCiuperci** conțin o pizza și metoda descriere() adaugă textul „cu peperoni” respectiv „cu ciuperci” la descrierea pizzei conținute. Prețul unei pizza care conține peperoni crește cu 2 Ron, cel cu ciuperci costa în plus 3 RON.
- Clasa **BasicPizza** reprezintă o pizza fără ciuperci și fără peperoni, metoda descriere() returnează denumirea pizzei. În pizzerie există 2 feluri de pizza de bază: Salami și Diavola, la prețul de 15 respectiv 20 RON.

Se cere:

1. Scrieți codul C++ doar pentru clasele **Pizza** și **PizzaCuPeperoni**. (0.75p)
2. Scrieți o funcție C++ care creează o comandă (returnează o lista de pizza) care conține: o pizza „Salami” cu ciuperci, o pizza „Salami” simplă, o pizza „Diavola” cu peperoni și ciuperci. (0.5p)
3. Scrieți programul principal care creează o comandă (folosind funcția descrisă mai sus), apoi tipărește descrierea și prețul pentru fiecare pizza în ordinea descrescătoare a prețurilor. (0.25p)

Obs. Creați doar metode și atrbute care rezultă din diagrama UML (adăugați doar lucruri specifice C++ ex: constructori). Nu adăugați câmpuri, metode, nu schimbați vizibilitatea, nu folosiți friend. Folosiți STL unde există posibilitatea.

Detalii barem: **1.5p** Polimorfism, **1p** Gestiona memoria, **1.5p** Restul

4 Definiți clasa Catalog astfel încât următoarea secvență C++ să fie corectă sintactic și să efectueze ceea ce indică comentariile. (2p)

```

void catalog() {
    Catalog<int> cat{ "OOP" }; //creaza catalog cu note intregi
    cat += 10; //adauga o nota in catalog
    cat = cat + 8 + 6;
    int sum = 0;
    for (auto n : cat) { sum += n; } //itereaza notele din catalog
    std::cout << "Suma note:" << sum << "\n";
}
  
```

3. class Pizza

{

private:

int pret;

public:

Pizza(int p) : pret(p){};

virtual string descriere() = 0;

int getPrete()

{

return pret;

}

virtual ~Pizza() = default;

};

class PizzaCuPeperoni : public Pizza

{

private:

Pizza * pizza;

public:

PizzaCuPeperoni(Pizza * p) : Pizza(p->getPrete()), pizza(p){};

string descriere override

{

return pizza->descriere() + " cu peperoni";

}

int getPrete override

{

return pizza->getPrete() + 2;

}

~PizzaCuPeperoni()

{ delete pizza; }

};

3.2. vector < Pizza*> f2()

{

vector < Pizza*> vec;

vec.push_back(new PizzaCuPeperoni(new BasicPizza("Salami", 15)));

vec.push_back(new BasicPizza("Salami", 15));

vec.push_back(new PizzaCuPeperoni(new BasicPizza("Dinval", 20)));

return vec;

};

3.3. int main()

{

vector < Pizza*> l = f2();

sort(l.begin(), l.end(), [] (Pizza * p1, Pizza * p2)

{

return p1->getPrete() > p2->getPrete();

});

for (auto el : v)

{ cout << el->descriere() << el->getPrete();

delete el; }

- Cut Dump Memory leaks;

return 0;

4 Definiți clasa Catalog astfel încât următoarea secvență C++ sa fie corectă sintactic și să efectueze ceea ce indica comentariile. (2p)

```
void catalog() {
    Catalog<int> cat{ "OOP" };//creaza catalog cu note intregi
    cat + 10; //adauga o nota in catalog
    cat = cat + 8 + 6;
    int sum = 0;
    for (auto n : cat) { sum += n; } //itereaza notele din catalog
    std::cout << "Suma note:" << sum << "\n";
}
```

```
class Catalog
{
private:
    string materie;
    vector<TElem> vec;
public:
    Catalog(string mat): materie(mat) {}
    Catalog& operator+(const TElem & elem)
    {
        vec.push_back(elem);
        return *this;
    }
    typename vector<TElem>::iterator begin()
    {
        return vec.begin();
    }
    typename vector<TElem>::iterator end()
    {
        return vec.end();
    }
};
```

(7)

1 Specificați și testați funcția: (1.5p)

```
int f(int x) {  
    if (x <= 0)  
        throw std::exception("Invalid argument!");  
  
    int rez = 0;  
    while (x)  
    {  
        rez = rez * 10 + x % 10;  
        x /= 10;  
    }  
    return rez;  
}
```

palindrom

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)  
#include <iostream>  
using namespace std;  
int except(bool thrEx) {  
    if (thrEx) {  
        throw 2;  
    }  
    return 3;  
}  
  
int main() {  
    try {  
        cout << except(1 < 1);  
        cout << except(true);  
        cout << except(false);  
    } catch (int ex) {  
        cout << ex;  
    }  
    cout << 4;  
    return 0;  
}
```

```
//2 b (0.5p)  
#include <iostream>  
using namespace std;  
class A {  
public:  
    A() {cout << "A" << endl;}  
    ~A() {cout << "~A" << endl;}  
    void print() {  
        cout << "print" << endl;  
    }  
};  
  
void f() {  
    A a[2];  
    a[1].print();  
}  
int main() {  
    f();  
    return 0;  
}
```

324

A A print ~A ~A

1 Specificați și testați funcția: (1.5p)

8

```
std::pair<int, int> f(std::vector<int> l) {
    if (l.size()<2) throw std::exception{};
    std::pair<int, int> rez{-1,-1};
    for (auto el:l) {
        if (el < rez.second) continue;
        if (rez.first < el) {
            rez.second = rez.first;
            rez.first = el;
        }else{
            rez.second=el;
        }
    }
    return rez;
}
```

cele mai mari două numere din vector
descrescător

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)
#include <iostream>
#include <vector>
struct A {
    A() {std::cout << "A";}
    virtual void print() {
        std::cout << "A";
    }
};
struct B : public A{
    B() { std::cout << "B"; }
    void print() override {
        std::cout << "B";
    }
};
int main() {
    std::vector<A> v; A
    v.push_back(A{}); A
    v.push_back(B{}); B
    for (auto& el : v) el.print();
    return 0;
}
```

```
//2 b (0.5p)
#include <iostream>
using namespace std;
class A {
    int x;
public:
    A(int x) : x{ x } {}
    void print(){cout<< x << endl;}
};
A f(A a) {
    a.print();
    a = A{ 10 };
    a.print();
    return a;
}
int main() {
    A a{ 4 };
    a.print();
    f(a);
    a.print();
}
```

AABAA

44104

9

1 Specificați și testați funcția: (1.5p)

```
bool f(int a) {
    if (a <= 0)
        throw std::exception("Illegal argument");
    int d = 2;
    while (d<a && a % d>0) d++;
    return d>=a;
}
```

a=prim $\Rightarrow 1$

a=impar $\Rightarrow 0$

a=par $\Rightarrow 0$

a<=2 $\Rightarrow 1$

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)
#include <iostream>
using namespace std;
class A {
public:
    A(){cout << "A()" << endl;}
    void print() {cout << "printA" << endl;}
};
class B: public A {
public:
    B(){cout << "B()" << endl;}
    void print() {cout << "printB" << endl;}
};
int main() {
    A* o1 = new A();
    B* o2 = new B();
    o1->print();
    o2->print();
    delete o1;delete o2;
    return 0;
}
```

```
//2 b (0.5p)
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> v;
    v.push_back(5);
    v.push_back(7);
    v[0] = 6;
    v.push_back(8);
    auto it = v.begin();
    it = it + 1;
    while (it != v.end()) {
        cout << *it << endl;
        it++;
    }
    return 0;
}
```

A() A() B() printA printB

7 8

10

1 Specificați și testați funcția: (1.5p)

```
bool f(int a) {
    if (a <= 1)           m prim sau nu
        throw "Illegal argument";
    int aux = 0;
    for (int i = 2; i < a; i++) {
        if (a % i == 0) {
            aux++;
        }
    }
    return aux == 0;
}
```

$a \leq 1 \Rightarrow \text{exc}$
 $a = \text{prim} \Rightarrow 1$

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)
#include <vector>
#include <iostream>
using namespace std;
class A {
public:
    virtual void f() = 0;
};
class B:public A{
public:
    void f() override {
        cout << "f din B";
    }
};
class C :public B {
public:
    void f() override {
        cout << "f din C";
    }
};
int main() {
    vector<A> v;
    B b;
    v.push_back(b);
    C c;
    v.push_back(c);
    for (auto e : v) { e.f(); }
    return 0;
}
```

```
//2 b (0.5p)
#include <iostream>
using namespace std;
class A {
public:
    A() {cout << "A" << endl;}
    ~A() {cout << "~A" << endl; }
    void print() {cout << "print" <<
endl;}
};
void f() {
    A a[2];
    a[0].print();
}
int main() {
    f();
    return 0;
}
```

eroare \Rightarrow A clasă abstractă

A A print ~A ~A

?

1 Specificați și testați funcția: (1.5p)

```
using namespace std;
#include <vector>
#include <string>
#include <algorithm>
#include <map>
vector<int> f(vector<int> l) {
    if (l.size() == 0)
        throw exception("Illegal argument");
    map<int, int> c;
    for (auto e : l) {
        c[e]++;
    }
    sort(l.begin(), l.end(), [&](int a, int b) {
        return c[a] > c[b];
    });
    return l;
}
```



2 Indicați rezultatul execuției pentru următoarele programe C++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)
#include <vector>
#include <iostream>
class A {
public:
    A() {
        std::cout << "A";
    }
    virtual void print() {
        std::cout << "printA";
    }
};
class B : public A {
public:
    B() {
        std::cout << "B";
    }
    virtual void print() {
        std::cout << "printB";
    }
};
int main() {
    std::vector<A> v;
    A a; A
    B b; B
    v.push_back(a);
    v.push_back(b);
    for (auto e : v) {e.print();}
    return 0;
}
```

```
//2 b (0.5p)
#include <iostream>
using namespace std;
class A {
public:
    A() {cout << "A" << endl;}
    ~A() {cout << "~A" << endl;}
    void print() {
        cout << "print" << endl;
    }
};
void f() {
    A a[2];
    a[1].print();
}
int main() {
    f();
    return 0;
}
```

A A print ~A ~A

3 Scrieți codul C++ ce corespunde diagramei de clase UML. (4p)

AABprintA printB

1 Specificați și testați funcția: (1.5p)

(12)

```
vector<int> f(int a) {
    if (a < 0)
        throw std::exception("Illegal argument");
    vector<int> rez;
    for (int i = 1; i <= a; i++) {
        if (a % i == 0) {
            rez.push_back(i);
        }
    }
    return rez;
}
```

rez - lista de divizori $a < 0 \Rightarrow$ exc

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)
#include <iostream>
using namespace std;
int except(int v) {
    if (v < 0) {throw 1; }
    else if (v > 0){
        throw std::exception ("A");
    }
    return 0;
}
int main(){
    try {
        cout << except(1 < 1); D
        cout << except(-5); A
        cout << except(5); -
    }catch (std::exception& e) {
        cout << "A";
    }catch (int x) {
        cout << "B";
    }
    cout << "C";
    return 0;
}
```

```
//2 b (0.5p)
#include <iostream>
using namespace std;
class A {
    int x;
public:
    A(int x) : x{ x } {}
    void print(){cout<< x << ",";};
};
A f(A a) {
    a.print();
    a = A{ 7 };
    a.print();
    return a;
}
int main() {
    A a{ 5 };
    a.print();
    f(a);
    a.print();
}
```

030

5,5,7,5

(13)

1 Specificați și testați funcția: (1.5p)

```
int f(int x) {
    if (x <= 0)
        throw std::exception("Invalid argument!");

    int rez = 0;
    while (x)
    {
        rez = rez * 10 + x % 10;
        x /= 10;
    }
    return rez;
}
```

palindrom

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)
#include <iostream>
using namespace std;
int except(bool thrEx) {
    if (thrEx) {
        throw 2;
    }
    return 3;
}

int main() {
    try {
        cout << except(1 < 1);
        cout << except(true);
        cout << except(false);
    } catch (int ex) {
        cout << ex;
    }
    cout << 4;
    return 0;
}
```

```
//2 b (0.5p)
#include <iostream>
using namespace std;
class A {
public:
    A() {cout << "A" << endl;}
    ~A() {cout << "~A" << endl;}
    void print() {
        cout << "print" << endl;
    }
};

void f() {
    A a[2];
    a[1].print();
}
int main() {
    f();
    return 0;
}
```

324

AA print ~A ~A

1 Specificați și testați funcția: (1.5p)

14

```
std::pair<int, int> f(std::vector<int> l) {
    if (l.size()<2) throw std::exception{};
    std::pair<int, int> rez{-1,-1};
    for (auto el:l) {
        if (el < rez.second) continue;
        if (rez.first < el) {
            rez.second = rez.first;
            rez.first = el;
        }else{
            rez.second=el;
        }
    }
    return rez;
}
```

cele mai mari două numere din vector
descrescător

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)
#include <iostream>
#include <vector>
struct A {
    A() {std::cout << "A";}
    virtual void print() {
        std::cout << "A";
    }
};
struct B : public A{
    B() { std::cout << "B"; }
    void print() override {
        std::cout << "B";
    }
};
int main() {
    std::vector<A> v;
    v.push_back(A{});
    v.push_back(B{});
    for (auto& el : v) el.print();
    return 0;
}
```

```
//2 b (0.5p)
#include <iostream>
using namespace std;
class A {
    int x;
public:
    A(int x) : x{x} {}
    void print(){cout<< x << endl;}
};
A f(A a) {
    a.print();
    a = A{ 10 };
    a.print();
    return a;
}
int main() {
    A a{ 4 };
    a.print();
    f(a);
    a.print();
}
```

AABA

44104

(15)

1 Specificați și testați funcția: (1.5p)

```

bool f(int a) {
    if (a <= 0)
        throw std::exception("Illegal argument");
    int d = 2;
    while (d<a && a % d>0) d++;
    return d>=a;
}

```

a=prim => 1

a=impar => 0
a=pair => 0
a<=2 => 1

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```

//2 a (1p)
#include <iostream>
using namespace std;
class A {
public:
    A(){cout << "A()" << endl;}
    void print() {cout << "printA" << endl;}
};
class B: public A {
public:
    B(){cout << "B()" << endl;}
    void print() {cout << "printB" << endl;}
};
int main() {
    A* o1 = new A();
    B* o2 = new B();
    o1->print();
    o2->print();
    delete o1;delete o2;
    return 0;
}

```

```

//2 b (0.5p)
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> v;
    v.push_back(5);
    v.push_back(7);
    v[0] = 6;
    v.push_back(8);
    auto it = v.begin();
    it = it + 1;
    while (it != v.end()) {
        cout << *it << endl;
        it++;
    }
    return 0;
}

```

A() A() B() printA printB

7,8

16

1 Specificați și testați funcția: (1.5p)

```

bool f(int a) {
    if (a <= 1)
        throw "Illegal argument";
    int aux = 0;
    for (int i = 2; i < a; i++) {
        if (a % i == 0) {
            aux++;
        }
    }
    return aux == 0;
}

```

mă prim sau nu

 $a \leq 1 \Rightarrow \text{exc}$ $a = \text{prim} \Rightarrow 1$

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```

//2 a (1p)
#include <vector>
#include <iostream>
using namespace std;
class A {
public:
    virtual void f() = 0;
};
class B:public A{
public:
    void f() override {
        cout << "f din B";
    }
};
class C :public B {
public:
    void f() override {
        cout << "f din C";
    }
};
int main() {
    vector<A> v;
    B b;
    v.push_back(b);
    C c;
    v.push_back(c);
    for (auto e : v) { e.f(); }
    return 0;
}

```

```

//2 b (0.5p)
#include <iostream>
using namespace std;
class A {
public:
    A() {cout << "A" << endl;}
    ~A() {cout << "~A" << endl; }
    void print() {cout << "print" << endl;}
    void f() {
        A a[2];
        a[0].print();
    }
    int main() {
        f();
        return 0;
    }
}

```

eroare \Rightarrow A clasa abs.

AA print nA nA

?

1 Specificați și testați funcția: (1.5p)

```
using namespace std;
#include <vector>
#include <string>
#include <algorithm>
#include <map>
vector<int> f(vector<int> l) {
    if (l.size() == 0)
        throw exception("Illegal argument");
    map<int, int> c;
    for (auto e : l) {
        c[e]++;
    }
    sort(l.begin(), l.end(), [&](int a, int b) {
        return c[a] > c[b];
    });
    return l;
}
```

17

2 Indicați rezultatul execuției pentru următoarele programe C++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)
#include <vector>
#include <iostream>
class A {
public:
    A() {
        std::cout << "A";
    }
    virtual void print() {
        std::cout << "printA";
    }
};
class B : public A {
public:
    B() {
        std::cout << "B";
    }
    virtual void print() {
        std::cout << "printB";
    }
};
int main() {
    std::vector<A> v;
    A a;
    B b;
    v.push_back(a);
    v.push_back(b);
    for (auto e : v) {e.print();}
    return 0;
}
```

```
//2 b (0.5p)
#include <iostream>
using namespace std;
class A {
public:
    A() {cout << "A" << endl;}
    ~A() {cout << "~A" << endl;}
    void print() {
        cout << "print" << endl;
    }
};
void f() {
    A a[2];
    a[1].print();
}
int main() {
    f();
    return 0;
}
```

AA print ~A ~A

3 Scrieți codul C++ ce corespunde diagramei de clase UML. (4p)

AAB printA printB

1 Specificați și testați funcția: (1.5p)

18

```
vector<int> f(int a) {
    if (a < 0)
        throw std::exception("Illegal argument");
    vector<int> rez;
    for (int i = 1; i <= a; i++) {
        if (a % i == 0) {
            rez.push_back(i);
        }
    }
    return rez;
}
```

rez - lista de divizori a < 0 \Rightarrow exc

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```
//2 a (1p)
#include <iostream>
using namespace std;
int except(int v) {
    if (v < 0) {throw 1; }
    else if (v > 0){
        throw std::exception ("A");
    }
    return 0;
}
int main(){
    try {
        cout << except(1 < 1);
        cout << except(-5);
        cout << except(5);
    }catch (std::exception& e) {
        cout << "A";
    }catch (int x) {
        cout << "B";
    }
    cout << "C";
    return 0;
}
```

```
//2 b (0.5p)
#include <iostream>
using namespace std;
class A {
    int x;
public:
    A(int x) : x{ x } {}
    void print(){cout<< x << ",";};
};
A f(A a) {
    a.print();
    a = A{ 7 };
    a.print();
    return a;
}
int main() {
    A a{ 5 };
    a.print();
    f(a);
    a.print();
}
```

0BC

5,5,7,5,

(19)

1 Specificați și testați funcția: (1.5p)

```

int f(int x) {
    if (x <= 0)
        throw std::exception("Invalid argument!");

    int rez = 0;
    while (x)
    {
        rez = rez * 10 + x % 10;
        x /= 10;
    }
    return rez;
}

```

palindrom

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```

//2 a (1p)
#include <iostream>
using namespace std;
int except(bool thrEx) {
    if (thrEx) {
        throw 2;
    }
    return 3;
}

int main() {
    try {
        cout << except(1 < 1);
        cout << except(true);
        cout << except(false);
    } catch (int ex) {
        cout << ex;
    }
    cout << 4;
    return 0;
}

```

```

//2 b (0.5p)
#include <iostream>
using namespace std;
class A {
public:
    A() {cout << "A" << endl;}
    ~A() {cout << "~A" << endl;}
    void print() {
        cout << "print" << endl;
    }
};

void f() {
    A a[2];
    a[1].print();
}
int main() {
    f();
    return 0;
}

```

324

AA print~A ~A

EXTRA

1 Specificați și testați funcția: (1.5p)

```

void f(vector<int>& l, int poz) {
    if (poz < 0 || poz >= l.size()) throw exception{};
    int a = 0;
    int b = l.size()-1;
    int nr = l[poz];
    while (a < b) {
        while (a < b && l[a] < nr) a++;
        while (b > a && l[b] > nr) b--;
        if (a < b) {
            swap(l[a], l[b]);
            if (l[a] == l[b] && l[b] == nr) a++;
        }
    }
}

```

poz invalidă \Rightarrow exc

a = 0, 1, 2, 3
b = 3
nr = 4

poz = 1
 $\ell = 1, 4, 2, 3$
 $a \leq b$

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea și motivul.

```

//2 a (1p)
#include <vector>
#include <iostream>
class A {
public:
    virtual void print() = 0;
};
class B : public A {
public:
    virtual void print() {
        std::cout << "printB";
    }
};
class C : public B {
public:
    virtual void print() {
        std::cout << "printC";
    }
};
int main() {
    std::vector<A> v;
    B b; C c;
    v.push_back(b);
    v.push_back(c);
    for (auto e : v) { e.print(); }
    return 0;
}

```

```

//2 b (0.5p)
void f(bool b) {
    std::cout << "1";
    if (b) {
        throw
    std::exception("Error");
    }
    std::cout << "3";
}
int main() {
    try {
        f(false);
        f(true);
        f(false);
    }
    catch (std::exception& ex) {
        std::cout << "4";
    }
    return 0;
}

//include <iostream>
using namespace std;

```

erorile \Rightarrow A clasei abs.

13/4

1 /* Functia punem in stanga elementului de pe pozitia poz numerele mai mici decat el, iar in dreapta sa numerele mai mari.

Param de intrare: l - vector de intregi, poz - intreg

Param de ieșire: nu avem

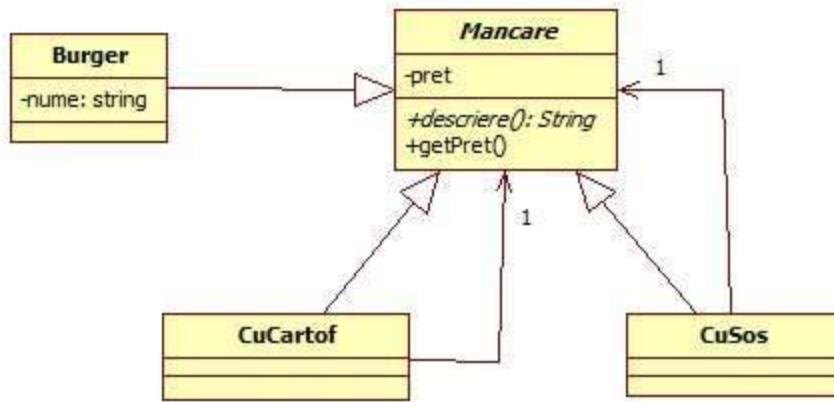
Excepție dacă poz nu este validă (<0 sau > lungimea vec)

```
*/ void test()
{
    vector<int> l {7,9,5,4,3};
    f(l,2);
    assert (l.at(0) == 3),
    assert (l.at(1) == 4),
    assert (l.at(2) == 5),
    assert (l.at(3) == 9),
    assert (l.at(4) == 7);

    try
    {
        f(l,-1);
        assert (false);
    }
    catch (exception)
    {
        assert (true);
    }

    try
    {
        f(l,5);
        assert (false);
    }
    catch (exception)
    {
        assert (true);
    }
}
```

3 Scrieți codul C++ ce corespunde diagramei de clase UML. (4p)



- Clasa abstractă **Mancare** are o metoda pur virtuală `descriere()`
- **CuCartof** și **CuSos** conțin o mâncare și metoda `descriere()` adaugă textul “cu cartof” respectiv “cu sos” la descrierea mâncării conținute. Prețul crește cu 3 RON pentru cartofi, mâncarea cu sos costa în plus 2 RON. Clasa **Burger** reprezintă un hamburger fără cartof și fără sos, metoda `descriere()` returnează denumirea hamburgerului.

Se cere:

- 1 Codul C++ doar pentru clasele: **Mancare**, **Burger**, **CuSos** (0.75) ✓
- 2 Scrieți o funcție C++ care returnează o listă de mâncăruri: un burger „McPuisor”, un burger „BigTasty” cu cartof și sos, un burger „Booster” cu cartof și un burger „Booster” cu sos (alegeți voi prețul de bază pentru fiecare mâncare). (0.5)
- 3 În programul principal creați o listă de mâncăruri (folosind funcția descrisă mai sus), apoi triplați descrierea și prețul pentru fiecare în ordinea descrescătoare a prețurilor. (0.25)
Creați doar metode și attribute care rezultă din diagrama UML (adăugați doar lucruri specifice C++ ex: constructori). Nu adăugați câmpuri, metode, nu schimbați vizibilitatea, nu folosiți friend și static. Folosiți STL unde există posibilitatea. Detalii barem: 1.5p Polimorfism, 1p Gestiona memoriei, 1.5p Restul (defalcat mai sus)

4 Definiți clasa Conferinta și Sesiune astfel încât următoarea secvență C++ să fie corectă sintactic și să efectueze ceea ce indică comentariile. (2p)

```

int main() {
    Conferinta conf;
    //add 2 attendants to "Artificial Inteligente" track
    conf.track("Artificial Inteligente") + "Ion Ion" + "Vasile Aron";
    //add 2 attendants to "Software" track
    Sesiune s = conf.track("Software");
    s + "Anar Lior" + "Aurora Bran";
    //print all attendants from group "Artificial Inteligente" track
    for (auto name : conf.track("Artificial Inteligente")) {
        std::cout << name << ",";
    }
    //find and print all names from Software track that contains "ar"
    vector<string> v = conf.track("Software").select("ar");
    for (auto name : v) { std::cout << name << ","; }
}

```

3. class Mamcare

```
{  
    private:  
        int pret;  
    public:  
        Mamcare (int pret): pret(pret) {}  
        virtual string descriere () = 0;  
        virtual int getPret();  
    }  
    ~Mamcare () = default;  
};
```

```
class CuSos : public Mamcare
```

```
{  
    private:  
        Mamcare * food;  
    public:  
        CuSos (Mamcare * f): Mamcare(f->getPret()), food(f) {}  
        string descriere () override  
    {  
        return food->descriere () + " cu sos ";  
    }  
    int getPret () override  
    {  
        return food->getPret () + 2;  
    }  
    ~CuSos() { delete food; }  
};
```

```
class Burger: public Mamcare
```

```
{  
    private:  
        string nume;  
    public:  
        Burger (int pret, string nume): Mamcare (pret), nume(nume) {}  
        string descriere () override  
    {  
        return nume + " ";  
    }  
};
```

3.2. vector < Mancare*> functie2()

```
{  
    vector < Mancare*> rez;  
    rez.push_back(new Burger(1, "McPurger"));  
    rez.push_back(new Salata((new Cusos((new Burger(1, "BigTasty))))));  
    rez.push_back((new Salata((new Burger(1, "Booster"))));  
    rez.push_back((new Cusos ((new Burger(1, "Booster"))));  
    return rez;  
}
```

3.3. int main()

```
{  
    vector < Mancare*> l = functie2();  
    sort(l.begin(), l.end(), [](Mancare* m1, Mancare* m2)  
    {  
        return m1->getPret() > m2->getPret();  
    });  
    for (const auto &m : l)  
    {  
        cout << m->descrivere() << m->getPret() << "\n";  
        delete m;  
    }  
    _GetDumpMemoryLeaks;  
    return 0;  
}
```

4 Definiți clasa Conferinta și Sesiune astfel încât următoarea secvență C++ să fie corectă sintactic și să efectueze ceea ce indică comentariile. (2p)

```
int main() {
    Conferinta conf;
    //add 2 attendants to "Artificial Inteligente" track
    conf.track("Artificial Inteligente") + "Ion Ion" + "Vasile Aron";
    //add 2 attendants to "Software" track
    Sesiune s = conf.track("Software");
    s + "Anar Lior" + "Aurora Bran";
    //print all attendants from group "Artificial Inteligente" track
    for (auto name : conf.track("Artificial Inteligente")) {
        std::cout << name << ",";
    }
    //find and print all names from Software track that contains "ar"
    vector<string> v = conf.track("Software").select("ar");
    for (auto name : v) { std::cout << name << ","; }
}
```

```
class Sesiune
{
public:
    vector<string> studenti;
    string cheie;
    Sesiune & operator+(const string &s)
    {
        studenti.push_back(s);
        return *this;
    }
    vector<string> & select(const string &s)
    {
        vector<string> rez;
        for (const auto& st: studenti)
        {
            if(st.find(s) != string::npos)
                rez.push_back(st);
        }
        return rez;
    }
    vector<string>::iterator begin()
    {
        return this->studenti.begin();
    }
    vector<string>::iterator end()
    {
        return this->studenti.end();
    }
}
```

```
class Conferinta
{
```

1 Specificati si testati functia: (1.5p)

```
using namespace std;
#include <vector>
#include <algorithm>
#include <stack>
vector<string> g(vector<string> l) {
    if (l.size() == 0) throw exception("Illegal argument");
    std::stack<string> st;
    for (auto& s : l) {
        st.push(s);
    }
    vector<string> r;
    while (!st.empty()) {
        r.push_back(st.top());
        st.pop();
    }
    return r;
}
```

$l = cabc$

$c bac$

palindrom cuvant

2 Indicati rezultatul executiei pentru urmatoarele programe c++. Daca sunt erori indicati locul unde apare eroarea si motivul.

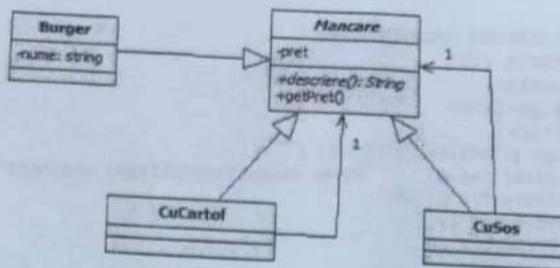
```
//2 a (1p)
#include <vector>
#include <iostream>
class A {
public:
    virtual void print() = 0;
};
class B : public A {
public:
    virtual void print() {
        std::cout << "printB";
    }
};
class C : public B {
public:
    virtual void print() {
        std::cout << "printC";
    }
};
int main() {
    std::vector<A> v;
    B b; C c;
    v.push_back(b);
    v.push_back(c);
    for (auto e : v) { e.print(); }
    return 0;
}
```

```
//2 b (0.5p)
void f(bool b) {
    std::cout << "1";
    if (b) {
        throw
            std::exception("Error");
    }
    std::cout << "3";
}
int main() {
    try {
        f(false);
        f(true);
        f(false);
    }
    catch (std::exception& ex) {
        std::cout << "4";
    }
    return 0;
}
```

erorile => A clasa abstrata

13/4

3 Scripteti codul C++ ce corespunde diagramei de clase UML. (2p)



- Clasa abstracta **Mancare** are o metoda pur virtuala descriere()
- **CuCartof** si **CuSos** contin o mancare si metoda descriere() adauga textul "cu cartof" respectiv "cu sos" la descrierea mancarii continue. Pretul creste cu 3 RON pentru cartofi, mancarea cu sos costa in plus 2 RON.
- Clasa **Burger** reprezinta o un hamburger fara cartof si fara sos, metoda descriere() returneaza denumirea hamburgerului.

Scripteti o functie C++ care returneaza o lista de mancaruri: un burger BigMac, un burger BigMac cu cartof si sos, un burger Zinger cu cartof si un burger Zinger cu sos (alegeti voi pretul de baza pentru fiecare mancare). In programul principal se creaza o lista de mancaruri (folosind functia descrisa mai sus), apoi tripariti descrierea si pretul pentru fiecare in ordinea descrescatoare a preturilor. Creati doar metode si atribute care rezulta din diagrama UML (adaugati doar lucruri specifice C++ ex: constructori). Implementati corect gestiunea memoriei. (2p)

4 Definiti clasa Carnet generala astfel incat urmatoarea secenta C++ sa fie corecta sintactic si sa efectueze ceea ce indica comentariile. (2p)

```
void anscolar() {
    Carnet<int> cat;
    cat.add("SDA", 9); //adauga nota pentru o materie
    cat.add("OOP", 7).add("FP", 10);
    cout << cat["OOP"]; //tiparesti nota de la materia data (7 la OOP);
    //removeLast() sterge ultima nota adaugata in carnet
    cat.removeLast().removeLast(); //sterge nota de la FP si OOP
    try{
        //se arunca exceptie daca nu exista nota pentru materia ceruta
        cout << cat["OOP"];
    }catch (std::exception& ex) {
        cout << "Nu exista nota pentru OOP";
    }
}
```

4 Definiti clasa Carnet generala astfel incat urmatoarea secevta C++ sa fie corecta
sintactic si sa efectueze ceea ce indica comentariile. (2p)

```
void anscolar() {
    Carnet<int> cat;
    cat.add("SDA", 9); //adauga nota pentru o materie
    cat.add("OOP", 7).add("FP", 10);
    cout << cat["OOP"]; //tipareste nota de la materia data (7 la OOP);
    //removeLast() sterge ultima nota adaugata in carnet
    cat.removeLast().removeLast(); //sterge nota de la FP si OOP
    try{
        //se arunca exceptie daca nu exista nota pentru materia ceruta
        cout << cat["OOP"];
    }catch (std::exception& ex) {
        cout << "Nu exista nota pentru OOP";
    }
}
```

class Carnet

{ private:

vector<pair<string, TElem>> lista;

public:

Carnet & add (const string & s, const TElem & elem)

{ lista.push_back(makepair(s, elem));
 return *this;
}

TElem operator [] (const string & s)

{ for (const auto & el : lista)
 if (el.first == s)
 return el.second;
 throw exception();
}

Carnet & removeLast()

{ lista.pop_back();
 return *this;
}

}

1. Specificati si testati.

```
vector<int> f(int a){  
    if(a<0) throw std::exception("Illegal argument");  
    vector<int> rez;  
    for(int i = 1;i<=a;i++){  
        if(a%i==0) rez.push_back(i);  
    }  
    return rez;  
}
```

lista de divizori

2. Ce rezulta in urma executiei codului. Daca sunt erori, semnalati.

a)

```
int except(int v) {  
    if (v < 0) throw 1;  
    else if (v > 0) throw exception("2");  
    return 0;  
}  
  
int main() {  
    try {  
        cout << except(1 < 1) << endl; 0  
        cout << except(-5) << endl;  
        cout << except(6) << endl;  
    }  
    catch (exception& e) {  
        cout << "2" << endl;  
    }  
    catch (int a) {  
        cout << "-" << endl;  
    }  
    cout << "+" << endl;  
    return 0;  
}
```

0-+

b)

```
class A {  
    int x;  
public:  
    A(int _a) : x{ _a } {};  
    void print() { cout << x << endl; }  
};  
  
A f(A a) {  
    a.print();  
    a = A{ 10 };  
    a.print();  
    return a;  
}  
  
int main() {  
    A a{ 4 };  
    a.print();  
    f(a);  
    a.print();  
    return 0;  
}
```

44104