1 Specificați si 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;
}
```

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

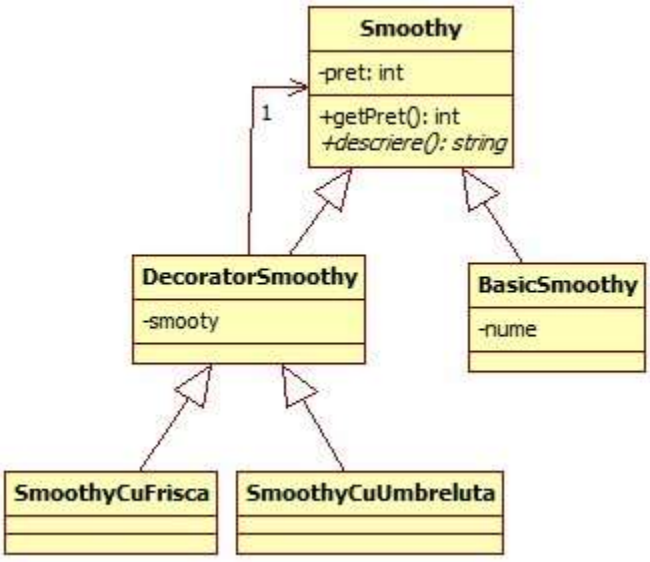
```
//2 a  (1p)
#include <iostream>
using namespace std;
int except(bool thrEx) {
	if (thrEx) {
		throw 2;
	}
	return 3;
}

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;
}
```

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



- Clasa abstracta **Smoothy** are o metodă pur virtuală descriere(). **DecoratorSmoothy** conține un smoothy, metodele descriere() si getPret() returnează descrierea si pretul smoothy-ului agregat.
- Clasele **SmoothyCuFrisca** si **SmoothyCuUmbreluta** adaugă textul "cu frisca" respectiv "cu umbreluta" la descrierea smoothy-ului conținut. Prețul unui smoothy care are frișca creste cu 2 Ron, cel cu umbrelușa costa in plus 3 RON.
- Clasa **BasicSmoothy** reprezintă un smoothy fără frișcă si fără umbrelușa, metoda descriere() returnează denumirea smothy-ului.

Se cere:

1 Codul C++ **doar pentru clasele**: **Smoothy**,**DecoratorSmoothy**, **SmoothyCuFrisca (0.75)**

2 Scrieți o funcție C++ care returnează o lista de smothy-uri: un smoothy de kivi cu frișcă si umbrelușă, un smoothy de căpșuni cu frișcă si un smoothy simplu de kivi. **(0.5p)**

3 Programul principal apelează funcția descrisa mai sus, apoi tipărește descrierea si prețul pentru fiecare smoothy in ordine alfabetica după descriere. **(0.25p)**

- Creați doar metode si atribute 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 exista posibilitatea.
- Detalii barem: **1.5p** Polimorfism, **1p** Gestiunea memoriei, **1.5p** Restul(defalcat mai sus)

4 Definiți clasa Geanta astfel încât următoarea secvență C++ sa fie corecta sintactic si 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";
	}
}
```

1 Specificați si testați funcția: (1.5p)

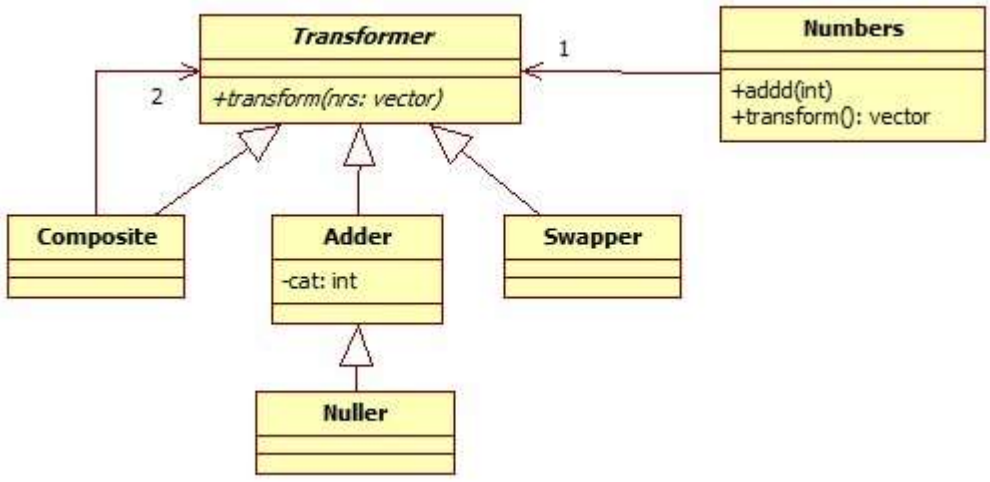
```
bool f(int a) {
     if (a <= 0)
          throw std::exception("Illegal argument");
     int d = 2;
     while (d<a && a % d>0) d++;
     return d>=a;
}
```

2 Indicați rezultatul execuției pentru următoarele programe c++. Dacă sunt erori indicați locul unde apare eroarea si 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;
}
```

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



- Clasa abstracta **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 **Nuller** înlocuiește numărul cu 0 daca in urma aplicări adunării numărul este > 10 sau lasă numărul ce rezulta in urma adunării. Clasa **Composite** in metoda transform() aplica succesiv cele doua transformări folosind **Transfromer**-ele agregate.
- Metoda transform() din clasa **Numbers** ordonează descrescător numerele adăugate cu add si apelează metoda transform(nrs) din Transformer-ul continut.

Se cere:

1 Codul C++ **doar pentru clasele**: **Transformer**,**Composite**, **Nuller (0.75p)**
2 Scrieți o funcție fiecare creează si returnează un obiect **Numbers** care compune un Nuller (cat=9) cu un Swapper compus cu un Adder (cat=3). **(0.5p)**
3 In funcția main apelați funcțiile de mai sus, adăugați cate 5 numere in cele doua obiecte **Numbers**. apoi apelați funcția transform pentru ambele. **(0.25p)**

- Creați 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. **Barem**: **1.5p** Polimorfism, **1p** Gestiunea memoriei, **1.5p** Defalcat mai sus

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
	//itereaza 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);
}
```

1 Specificați si testați funcția: (1.5p)

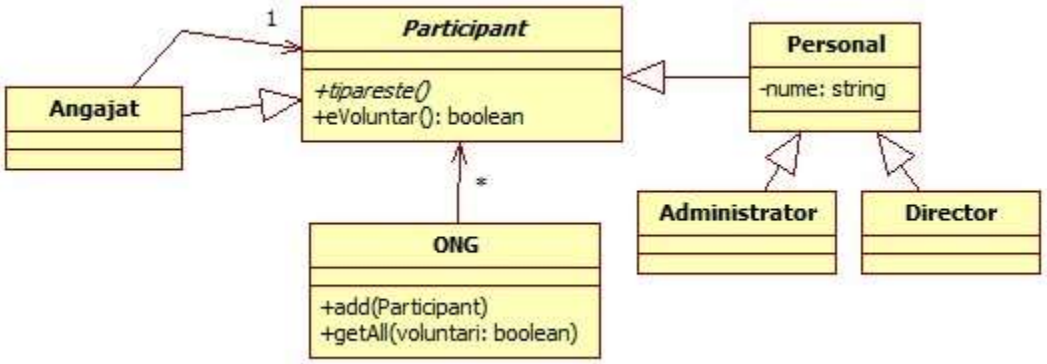
```
using namespace std;
#include <vector>
#include <string>
#include <algorithm>
#include <map>
vector<int> f(vector<int> l) {
	if (l.size() == 0)
		throw exception("Illegal argument");
	map<int, int> c;
	for (auto e : l)  {
		c[e]++;
	}
	sort(l.begin(), l.end(), [&](int a, int b) {
		return c[a] >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

2 Indicați rezultatul execuției pentru următoarele programe c++. Daca sunt erori indicați locul unde apare eroarea si 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;
}
```

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



- Clasa abstracta **Participant** are o metoda pur virtuala *tipareste*.
- Metoda *tipareste* din clasa **Personal** tipărește numele persoanei.
- Clasa **Administrator** si **Director** pe lângă ce tipărește clasa de baza mai tipărește si cuvântul “Administrator” respectiv “Director” .
- Clasa **Angajat** tipărește, pe lângă ce tipărește personalul agregat de el, si 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 (in funcție de parametru). Implicit toți participanții sunt voluntari daca 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 si returnează un obiect **ONG** si adaugă următorii participanți (alegeți voi numele pentru fiecare): un administrator voluntar, un administrator angajat, un director voluntar si un director angajat. **(0.5p)**

3 In funcția main tipăriți separat angajații si voluntarii din ONG. **(0.25p)**

Creați 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 exista posibilitatea.

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

4 Definiți clasa Cos generală astfel încât următoarea secvența 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)
}
```

1 Specificați si testați funcția: **(1.5p)**

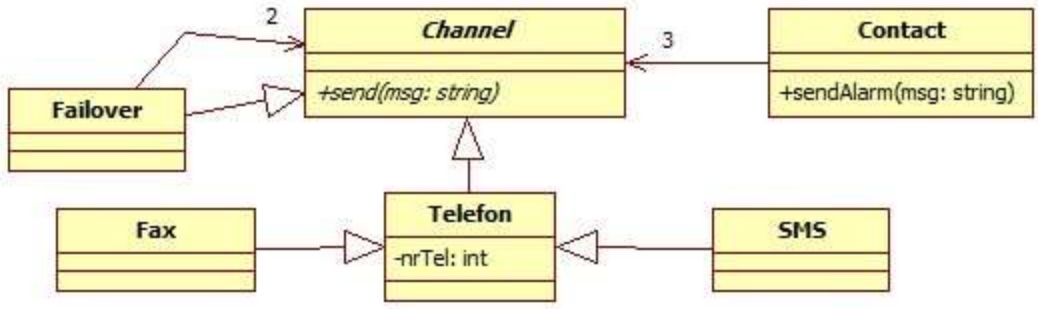
```
bool f(int a) {
	if (a <= 1)
		throw "Illegal argument";
	int aux = 0;
	for (int i = 2; i < a; i++) {
		if (a % i == 0) {
			aux++;
		}
	}
	return aux == 0;
}
```

2 Indicați rezultatul execuției pentru următoarele programe c++. Daca sunt erori indicați locul unde apare eroarea si 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;
}
```

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 in când (in funcție de un număr aleator generat) aruncă excepție std::exception indicând ca linia este ocupată.
- Clasa **Fax** si **SMS** încearcă sa apeleze numărul de telefon si in caz de succes tipărește "sending fax" respectiv "sending sms". Clasa **Failover** încearcă sa trimită mesajul pe primul canal, dacă trimiterea eșuează (este ocupat) atunci încearcă trimiterea pe canalul secundar.
- Metoda *sendAlarm* din clasa **Contact**, încearcă sa trimită repetat mesajul pe cele 3 canale conținute pe rând până reușește trimiterea (găsește o linie care nu este ocupat).

Se cere:

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

2 Scrieți o funcție C++ care creează si 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 In funcția main apelați funcția de mai sus si trimiteți 3 mesaje. **(0.25p)**

- Creați 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 exista posibilitatea.

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

4 Definiți clasa Expresie generală astfel încât următoarea secvență C++ sa fie corecta sintactic si 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)
}
```

1 Specificați si testați funcția: (**1.5p**)

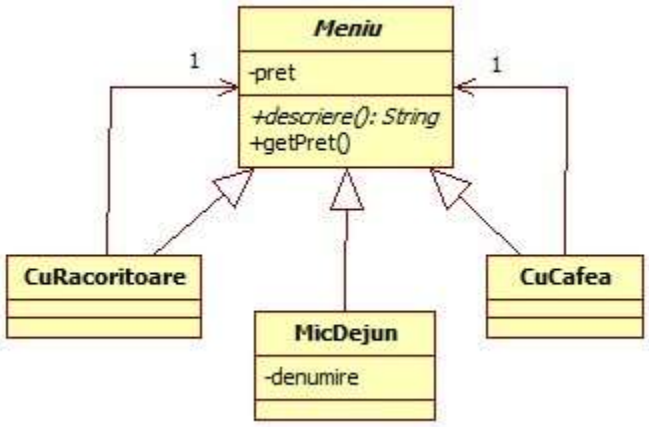
```
std::pair<int, int> f(std::vector<int> l) {
	if (l.size()<2) throw std::exception{};
	std::pair<int, int> rez{-1,-1};
	for (auto el:l) {
		if (el < rez.second) continue;
		if (rez.first < el) {
			rez.second = rez.first;
			rez.first = el;
		}else{
			rez.second=el;
		}
	}
	return rez;
}
```

2 Indicați rezultatul execuției pentru următoarele programe c++. Daca sunt erori indicați locul unde apare eroarea si 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();
}
```

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



- Clasa abstracta **Meniu** are o metoda pur virtuala descriere()
- **CuRacoritoare** si **CuCafea** conțin un meniu si metoda descriere() adaugă textul "cu racoritoare" respectiv "cu cafea" la descrierea meniului conținut. Prețul unui meniu care conține răcoritoare creste cu 4 Ron, cel cu cafea costa in plus 5 RON.
- Clasa **MicDejun** reprezintă un meniu fără răcoritoare si fără cafea, metoda descriere() returnează denumirea meniului. In restaurant pizzerie exista 2 feluri de mic dejun: Ochiuri si 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 lista de meniuri: un meniu cu Omleta cu răcoritoare si cafea, un meniu cu Ochiuri si cafea, un meniu cu Omleta. **(0.5p)**
3 In programul principal se creează o comanda (folosind funcția descrisa mai sus), apoi se tipărește descrierea si prețul pentru fiecare pizza in ordinea descrescătoare a preturilor. **(0.25p)**

- Creați doar metode si atribute 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** Gestiunea memoriei, **1.5p** Restul(defalcat mai sus)

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 valuarea 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;
}
```