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.

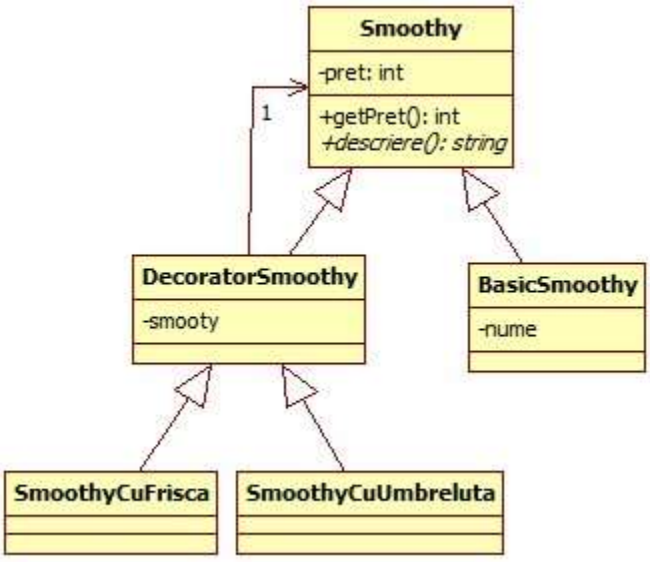
```
//2 a  (1p)
#include <iostream>
using namespace std;
int except(bool thrEx) {
	if (thrEx) {
		throw 2;
	}
	return 3;
}

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";
	}
}
```