Tema 2

Termen de predare: laboratorul din săptămâna a V-a

Obiective:

- Relația de *compoziție* (HAS-A) : sub-obiecte dependente
  - Tare: obiectul posedă sub-obiectele
  - Slabă: obiectul nu posedă sub-obiectele
- Relația de *moștenire* (IS-A)
- Ordinea de apel pentru constructori/destructori
- Liste de inițializare pentru constructori
- Vizibilitatea membrilor din clasa de bază în clasa derivată
- Redefinirea metodelor
- Polimorfism static: supraîncărcarea operatorilor
- Fluxuri de intrare/iesire: utilizarea fișierelor
- Utilizare STL: *list*, *string*, *stack*

Cerințe:

- Problemele trebuie prezentate pâna în săptămâna a 5-a inclusiv
- Fiecare problemă este notată cu maxim 10 puncte
- Toate problemele sunt obligatorii

Probleme:

1. Implementați o clasă *ArboreBinar* care să permită evaluarea unor expresii aritmetice elementare.

   - Expresiile pot conține numere raționale pozitive, operatorii binari +,-,*,/ și paranteze, separate cu spații; de exemplu, “100 + .2 * (3 + 9) / 4”.
   - Pentru evaluare, o expresie va fi transformată în forma postfixată.
   - Cu ajutorul formei postfixate, se va construi un arbore binar. Evaluarea expresiei inițiale se va realiza prin intermediul acestui arbore.

```
// file: ArboreBinar.h

#ifndef ARBOREBINAR
#define ARBOREBINAR

#include <vector>
#include <string>

class ArboreBinar
{
	char   cOperator;
	double dNumber;
	std::vector<ArboreBinar*> fii;
	bool valid;

	ArboreBinar(double); // constructor pentru initializarea unui nod frontiera
	ArboreBinar(char);   // constructor pentru initializarea unui nod intern


	// metoda pentru transformarea in forma postfixata a unei expresii
	static std::string InfixToPostfix( const std::string& expresie );
```

```
public:

	// constructor pentru initializarea nodului radacina
	ArboreBinar( const std::string& expresie );

	// returneaza true daca arborele a fost construit (expresia era corecta)
	bool EsteValid();

	// returneaza expresia rezultata in urma parcurgerii arborelui
	// (inordine, preordine, postordine)
	std::string Parcurgere( int nCodParcurgere );

	// returneaza rezultatul evaluarii arborelui
	double Evalueaza();
};

#endif

// file: ArboreBinar.cpp
#include "ArboreBinar.h"
#include <iostream>
#include <sstream>
#include <stack>

string ArboreBinar::InfixToPostfix(const string& expresie)
{
	// implementarea unui algoritm de tip shunting-yard

	stringstream stream;		// flux deschis simultan pentru scriere si citire
	stream << expresie;		// scriem expresia in flux

	stack<string> aStack;
	string postExpresie;
	char chSpace = ' ';

	while ( stream )
	{
		string strNumar;
		string strOperator;

		char aChar = stream.peek();		// ce caracter urmeaza in flux?

		switch ( aChar )
		{
			case ' ' :	stream.ignore();	// "sarim" peste spatiu
					continue;

			case '.' :
			case '0' :
			case '1' :
			case '2' :
			case '3' :
			case '4' :
			case '5' :
			case '6' :
			case '7' :
			case '8' :
			case '9' :	stream >> strNumar;		// citim numarul (ca sir)
					postExpresie += strNumar;	// scriem numarul
					postExpresie += chSpace;
					break;

			case '+' :
			case '-' :
```

```
                stream >> strOperator;      // citim operatorul
                while ( ( ! aStack.empty() )
                        &&
                        ( ( aStack.top() == "*" ) || ( aStack.top() == "/" )
                          ||
                          ( aStack.top() == "+" ) || ( aStack.top() == "-" )
                        )
                      )
                {
                        postExpresie += aStack.top();      // scriem operatorul
                        postExpresie += chSpace;
                        aStack.pop();                      // eliminam din stiva
                }

                aStack.push( strOperator );        // introducem operatorul in stiva
                break;

                case '*' :
                case '/' :

                stream >> strOperator;             // citim operatorul

                while ( ( ! aStack.empty() )
                        &&
                        ( ( aStack.top() == "*" ) || ( aStack.top() == "/" ) )
                      )
                {
                        postExpresie += aStack.top();      // scriem operatorul
                        postExpresie += chSpace;
                        aStack.pop();                      // eliminam din stiva
                }
                aStack.push( strOperator );        // introducem operatorul in stiva
                break;

                case '(' :    stream.ignore();
                              aStack.push( "(" );  // introducem '(' in stiva
                              break;

                case ')' :

                stream.ignore();
                while ( ( ! aStack.empty() ) && ( aStack.top() != "(" ) )
                {
                        postExpresie += aStack.top();      // scriem in post-expresie
                        postExpresie += chSpace;
                        aStack.pop();                      // eliminam din stiva
                }
                aStack.pop();                              // eliminam '(' din stiva
                break;

                case EOF : continue;

                default  : return "Eroare la parsarea expresiei!";
        }
}

// golim stiva
while ( ! aStack.empty() )
{
        postExpresie += aStack.top();
        postExpresie += chSpace;
        aStack.pop();
}

return postExpresie;
```

}

Scrieţi un program în care să citiţi o expresie şi să o evaluaţi prin intermediul acestei clase!

2. Un *Autoturism* este compus dintr-un *Motor*, un *Electromotor* şi un *SistemPornire*. Motorul poate porni şi se poate opri; un motor nu poate fi pornit decat cu ajutorul unui Electromotor asociat. Sistemul de pornire (poate fi oricât de complex, putând include baterie, cabluri, etc) controlează motorul şi electromotorul, trimiţându-le comenzi de pornire şi oprire. Autoturismul poate fi pornit (operaţie mai complexă, ce implică, pe lângă pornirea motorului, şi alte manevre precum aprinderea luminilor de poziţie), condus la destinaţie şi parcat (stingerea luminilor de pozitie, oprirea motorului, etc). Ierahia de clase corespunzătoare poate arăta astfel:

```
class Motor
{
        public:
        void start();
        void stop();
};

class Electromotor
{
        public:
        void start(Motor&);
};

class SistemPornire
{
        Motor& m;
        Electromotor& e;

        public:
        SistemPornire (Motor&,  Electromotor&);

        void porneste_motor();
        void opreste_motor();
};

class Autoturism
{
        Electromotor electromotor;      //autoturismul poseda un electromotor
        Motor motor;                    //autoturismul poseda un motor
        SistemPornire sistem_pornire;   //autoturismul poseda un sistem de pornire

        public:
        Autoturism();

        void porneste_autoturism();
        void condu_la_destinatie();
        void parcheaza_autoturism();
};
```

- Implementaţi metodele propuse şi simulaţi într-un program conducerea unui autoturism.
- Adăugaţi o nouă clasă, *SistemPornireCuPreîncălzire*, care, inainte de a porni motorul, incălzeşte carburantul (pentru a uşura pornirea iarna). Modificaţi clasa Autoturism a.î. să utilizeze noul sistem de pornire şi exemplificaţi într-un program.
- Din cauza costurilor suplimentare, se hotărăşte ca SistemPornire să rămână în dotarea standard a unui autoturism, iar noul sistem de pornire să fie montat doar la cererea cumpărătorului. Modificaţi corespunzător definiţia clasei Autoturism şi exemplificaţi într-un program.

- Evident, un autoturism va fi dotat şi cu *Anvelope*; acestea pot fi de vară sau de iarnă şi, spre deosebire de celelalte componente, pot fi schimbate oricând între două călătorii. Modificaţi programele anterior realizate a.î. metoda *condu_la_destinatie()* să afişeze şi tipul de anvelopă de pe fiecare roată.

3. O *Bibliotecă* este compusă din *Publicaţii*. Fiecare componentă este identificată în cadrul bibliotecii printr-o *cotă* unică; evident, biblioteca poate conţine mai multe exemplare ale aceleiaşi publicaţii. O publicaţie este descrisă de un *titlu*, o *editură*, un *an al apariţiei* şi un *tiraj*. Există două tipuri de publicaţii: *Cărţi* şi *Reviste*. O carte este descrisă suplimentar de un *autor* (sau un grup de autori). O revistă este descrisă suplimentar de un *număr* şi de o *frecvenţă de apariţie*. Scrieţi un program care să simuleze funcţionarea unei biblioteci, implementând metode pentru:

   - adăugarea unei noi publicaţii în bibliotecă;
   - eliminarea unei publicaţii din bibliotecă (prin distrugere naturală);
   - afişarea descrierii complete a unei publicaţii;
   - căutarea după titlul, după cotă, după autor, etc;
   - raportarea numărului de publicaţii/reviste/cărţi distincte aflate în evidenţa bibliotecii;
   - salvarea conţinutului bibliotecii într-un fişier;
   - restaurarea conţinutului bibliotecii dintr-un fişier.

4. O *Localitate* este o unitate administrativ-teritorială în care trăieşte o comunitate de *Persoane*. Fiecare localitate are un *nume*, este identificată unic printr-un *cod poştal* şi este condusă de către una dintre persoanele care trăieşte în respectiva localitate şi care îndeplineşte funcţia de *primar*. *Judeţul* este un grup de localităţi. Populaţia unei localităţi poate scădea (prin decesul persoanelor) sau poate creşte (prin naşterea de noi persoane). Dacă populaţia unei localităţi ajunge să fie mai mare decât 10, localitatea respectivă devine *oraş*; dacă populaţia unei localităţi ajunge 0, respectiva localitate dispare din evidenţa judeţului. Un oraş nu poate redeveni simplă localitate prin scăderea populaţiei. Fiecare persoană are un *nume*, o *vârstă* şi este identificată unic prin *CNP*. O persoană majoră poate ocupa funcţia de primar şi poate vota pentru alegerea primarului sau la referendum. O persoană se poate muta în altă localitate (caz în care, eventual, pierde funcţia de primar). Scrieţi un program care să simuleze evoluţia vieţii într-un judeţ:

   - citirea judeţului dintr-un fişier;
   - salvarea judeţului intr-un fişier;
   - naşterea şi decesul persoanelor;
   - mutarea persoanelor în altă localitate;
   - alegerea primarului dintre persoanele majore (cu majoritate simplă);
   - creşterea populaţiei unei localităţi până la transformarea ei în oraş;
   - dispariţia unei localităţi;
   - organizarea şi afişarea rezultatelor unui referendum la nivel de localitate/judeţ.