LABORATORUL 2

Functii și clase prieten. Listă de inițializare. Metode și câmpuri statice.

1 Funcții și clase prieten

Funcțiile prieten sunt funcții care au access la câmpurile private și protected ale unei clase. Pentru a declara o funcție prieten, e suficient să declarăm signatura funcției în momentul declarării clasei atașând cuvântul cheie *friend* în față. Sintaxă:

```
class <nume_clasa> {
    /*
        definitii campuri
    */
    public:
    /*
        definitii metode
    */
    friend <tip_retur> <nume_functie> (<lista_parametri>);
};
```

Putem declara și *clase prieten* pentru o clasă. Ca în cazul funcțiilor, clasele declarate ca prieten au acces la câmpurile private și protected ale clasei declarate. Pentru a declara o clasă prieten trebuie să definim signatura clasei care umează să fie definită ca prieten. Sintaxă:

```
// signatura clase care umeaza sa fie declarata ca prieten
class <nume_clasa_1>;

class <nume_clasa_2> {
    /*
        definitii campuri
    */
    public:
        /*
            definitii metode
        */
        friend class <nume_clasa_1>;
};
```

2 Listă de inițializare

Atunci când implementăm un constructor, imediat după lista de parametri, putem enumera felul în care se inițializează câmpurile clasei. În lista de inițializare putem specifica cum să apelăm constructorul pentru câmpurile alocate static, inițializa câmpurile const sau specifica apeluri explicite ale constructorilor clasei de bază (la moștenire). Lista de inițializare poate fi folosită atât la declarații inline implicite cât și în cazul declarărilor normale. Sintaxa:

```
class <nume_clasa> {
    /*
        definitii campuri
    */
    public:
        /*
            definitii metode
        */
        <nume_clasa> (<lista_parametri>);
};


<nume_clasa>::<nume_clasa>(<lista_parametri>) : <lista_initializare> {
    // implementare constructor
}
```

Exemplu:

```
    class A {
        int x;
        string s;
        double *d;
        public:
           A () : x(1), s("202345"), d(NULL) {}
    };
```

3 Metode și câmpuri statice

Câmpurile statice sunt proprietăți care sunt împărțite de toate obiectele unei clase. Un câmp static este inițializat cu valoarea default pentru tipul de date declarat, dacă altă inițializare nu este specificată. Pentru a declara un câmp static trebuie folosit cuvântul cheie *static*. Sintaxă:

```
class <nume_clasa> {
    /*
        definitii campuri
    */
    static <tip_date> <nume_camp>;
    public:
    /*
        definitii metode
    */
```

```
};

// initializare camp static
<tip_date> <nume_clasa>::<nume_camp> = <valoare_initiala>;
```

O clasă poate avea și *metode statice*. Metodele statice nu au pointerul **this** și pot accesa doar câmpuri statice și pot apela doar alte metode statice ale clase. Metodele statice nu pot fi apelate folosind obiect, singura modalitate de apelare fiind folosirea operatorului rezoluție de scop:

```
<nume_clasa>::<nume_metoda>(parametri);
```

Pentru a declara o metodă statică e sufcient să adăugă cuvântul cheie static în fața signaturii metodei. Sintaxă:

```
class <nume_clasa> {
    /*
        definitii campuri
    */
    public:
    /*
        definitii metode
    */
    static <tip_retur> <nume_metoda>(lista_parametri);
};
```

Exerciții

Toate clasele de mai jos vor fi declarate fișierul header (.h) și implementate în fișierul sursă (.cpp). Fișierele în care o clasă este declarată și implementată trebuie să aibă numele clasei. În cazul constructorilor, se va folosi lista de inițializare peste tot unde este posibil.

1. Declarați clasa **Node** care are următoare proprietăți:

- **value** de tip **int** , valoarea stocată în nod;
- **right** de tip **Node*** , adresă către nodul din dreapta;
- **left** de tip **Node*** , adresă către nodul din stânga;

Implementați toate tipurile de constructori.

2. Declarați clasa **BST** (arbore binar de căutare) cu următoarele proprietăți:

- **root** de tip **Node*** , rădăcina arborelui;
- **count** de tip **unsigned** , numărul de elemente din arbore;

Implementați următorii constructori:

- **BST()** care inițializează câmpul **root** cu valoarea **NULL** ;

- **BST(BST&)** - constructorul de copiere;

3. Declarați clasa **BST** ca prieten la clasa **Node** .

4. Declarați și implementați metode de inserare (**void insert(int)**) și ștergere (**void remove(int)**) pentru clasa **BST** .

5. Declarați și implementați în clasa **Node** câmpul static **objects** de tip **int** care numară câte obiecte de tip **Node** au fost instanțiate.

6. Declarați și implementați o funcție prieten la clasa **BST** care afișează un obiect de tip **BST** în inordine (**void printTree(BST);**).

7. Declarați și implementați metode statice în clasa **BST** pentru traversarea arborelui în preordine și postordine. Metodele vor întoare pointer către un array care conține elementele în ordinea dată de parcurgere (**int* preorder(BST)** și **int* postorder(BST)**).

8. Declarați și implementați metoda statică **print** în clasa **BST** care primește un parametru de tip **BST** și un parametru de tip **int** . Dacă parametrul de tip **int** are valoarea 0 atunci este afișat arborele în preordine, altfel este afișat arborele în postordine (**void print(BST, int)**).

9. Declarați și implementați metoda **height** care întoarce înălțimea arborelui (**int height()**).