Laborator 8

Interpretarea arborilor binari în Haskell/ML

Laboratorul 8 constă în crearea unui analizor lexico-sintactic pentru procesarea arborilor binari de căutare în sintaxă Haskell și, comparativ, în sintaxă ML. Partea de ML va apărea pe fundal color (albastru), iar pentru implementare vom merge pe una dintre cele două sintaxe. Vom avea în vedere citirea arborilor și executarea unor operații pe arbori: **insert** (inserarea unui nod într-un arbore) și **count** (contorizarea nodurilor, diferite de frunzele vide Lf, dintr-un arbore).

Exemple de input în Haskell:

```
Node Lf 2 Lf
> Node Lf 2 Lf

Node (Node Lf 2 Lf) 10 (Node Lf 12 Lf)
> Node (Node Lf 2 Lf) 10 (Node Lf 12 Lf)

count (insert 16 (Node (Node Lf 2 Lf) 15 Lf))
> 3

insert (count (Node Lf 17 (Node Lf 24 Lf))) (insert 12 (Node Lf 10 Lf))
> Node (Node Lf 2 Lf) 10 (Node Lf 12 Lf)
```

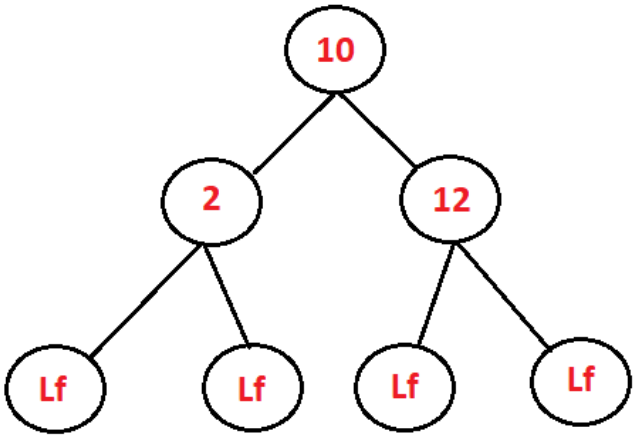
Exemple de input în ML:

```
Node(2, Lf, Lf)
> Node(2, Lf, Lf)

Node(10, Node(2, Lf, Lf), Node(12, Lf, Lf))
> Node(10, Node(2, Lf, Lf), Node(12, Lf, Lf))

count(insert(16, Node(15, Node(2, Lf, Lf), Lf)))
> 3

insert(count(Node(17, Lf, Node(24, Lf, Lf))), insert(12, Node(10, Lf, Lf)))
> Node(10, Node(2, Lf, Lf), Node(12, Lf, Lf))
```

După cum se poate observa în exemple, sintaxa Haskell a unui arbore binar este următoarea: `Node (Tree Int) Int (Tree Int)`

Iar sintaxa ML a unui arbore binar este: `Node (int, int Tree, int Tree)`

Să luăm ca exemplu arborele binar din imaginea următoare:



Reprezentarea arborelui în sintaxă Haskell este următoarea:

```
Node (Node Lf 2 Lf) 10 (Node Lf 12 Lf)
```

Iar reprezentarea în sintaxă ML este:

```
Node(10, Node(2, Lf, Lf), Node(12, Lf, Lf))
```

Pentru a reprezenta în memorie arborii binari de căutare, vom crea o structură cu trei câmpuri: unul pentru cheia nodului (de tip int) și două pentru fiii nodului (pointeri la subarborele stâng și la subarborele drept).

```
typedef struct _node {
      int key;
      struct _node *left, *right;
} node;
```

Pe stiva de valori vor coexista două tipuri: **int**, pentru numerele citite din input și pentru cheile nodurilor, și o structură de tip **arbore**, pentru a reduce numerele din input la arbori în forma potrivită (reducând partea dreaptă a producțiilor la partea stângă). Astfel, **yylval** va fi un union cu două câmpuri:

```
%union {
      int ival;
      struct _node *btree;
}
```

În ceea ce privește setul de producții, vom avea un set de reguli (**expr**) care descriu inputul. Putem separa instrucțiunile din input în două categorii, în funcție de tipul rezultatului. Citirile arborilor și comanda **insert** vor returna arbori, în vreme ce comanda **count** returnează un

număr întreg. Astfel, reies alte două seturi de reguli, unul pentru situațiile din care rezultă un arbore (**t_expr**) și unul pentru cele din care rezultă numere întregi (**i_expr**). Cel mai important set de reguli este cel care descrie construirea unui arbore binar (**tree**), cu ajutorul celor doi constructori de date Haskell/ML pentru arbori binari: constructorul Node cu trei argumente și constructorul Lf cu zero argumente.

Setul de producții pentru sintaxă Haskell:

```
expr : i_expr
     | t_expr
     ;

i_expr : COUNT t_expr
        |'(' i_expr ')'
        | NUMBER
        ;

t_expr : INSERT i_expr t_expr
       | '(' t_expr ')'
       | tree
       ;

tree : NODE tree NUMBER tree
     | '(' tree ')'
     | LF
     ;
```

Setul de producții pentru sintaxă ML:

```
expr : i_expr
     | t_expr
     ;
i_expr : COUNT '(' t_expr ')'
       | NUMBER
       ;
t_expr : INSERT '(' i_expr ',' t_expr ')'
       | tree
       ;
tree : NODE '(' NUMBER ',' tree ',' tree ')'
     | LF
     ;
```

Exerciții propuse:

1. Implementați o funcție pentru căutarea unui nod într-un arbore.
   Exemple în Haskell:
   find 12 (Node (Node Lf 2 Lf) 10 (Node Lf 12 Lf))
   > true
   find 17 (Node (Node Lf 2 Lf) 10 (Node Lf 12 Lf))
   > false

   Exemple în ML:

   ```
   find(12, Node(10, Node(2, Lf, Lf), Node(12, Lf, Lf)))
   > true
   find(17, Node(10, Node(2, Lf, Lf), Node(12, Lf, Lf)))
   > false
   ```

2. Implementați o funcție pentru ștergerea unui nod dintr-un arbore. Verificați întâi dacă nodul căutat există în arbore.
   Exemplu în Haskell:
   delete 12 (Node (Node Lf 2 Lf) 10 (Node Lf 12 Lf))
   > Node (Node Lf 2 Lf) 10 Lf

   Exemplu în ML:

   ```
   delete(12, Node(10, Node(2, Lf, Lf), Node(12, Lf, Lf)))
   > Node(10, Node(2, Lf, Lf), Lf)
   ```

3. Creați o funcție pentru a verifica dacă un arbore binar este echilibrat.
   Exemple în Haskell:
   balanced (Node (Node Lf 2 Lf) 10 (Node Lf 12 Lf))
   > true
   balanced (Node (Node Lf 2 Lf) 10 Lf)
   > false

   Exemple în ML:

   ```
   balanced(Node(10, Node(2, Lf, Lf), Node(12, Lf, Lf)))
   > true
   balanced(Node(10, Node(2, Lf, Lf), Lf))
   > false
   ```