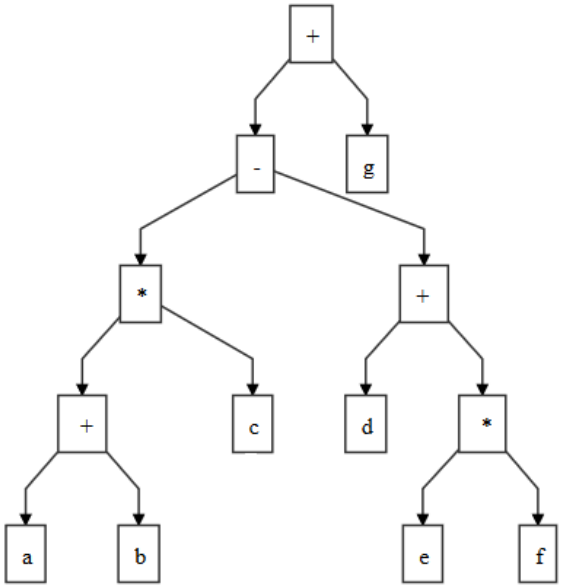
SDA - Seminar 7

- Stadiul proiectului pentru cei care au proiecte cu arbori

1. Să se construiască arborele binar atașat unei ezpresiiaritmetice conținând operatorii +, -, *, /, pornind de la forma poloneză postfixată.
   Ex: (a + b)* c – (d + e * f) + g => FPP: ab+c*def*+-g+
   Arborele atașat este:

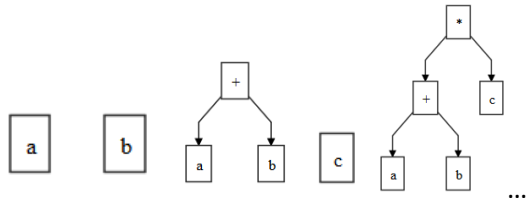


Dacă parcurg arborele în postordine obțin expresia în forma postfixată.

Algoritm:

1. Folosesc o stivă care conține adresele nodurilor din arbore.
2. Construiesc de jos în sus
3. Se parcurge expresia în FPP
4. Întâlnesc un operand -> îl adaug în stivă
5. Întâlnesc un operator ->
   a. Scot ultimul element din stivă – fiul drept
   b. Scout penultimul element din stivă – fiul stâng
   c. Creez un nod cu informație operator și fiul drept si stâng
   d. Adaug nodul creat în stivă
6. Rădăcina arborelui va fi ultimul element din stivă

Stiva:



...

Presupunem că avem arbore cu reprezentare înlănțuită cu alocare dinamică

Nod:
e:TElement
st, dr: ↑Nod

AB:
răd: ↑Nod

Stiva va avea elemente de tip Nod și vom folosi operațiile stivei:

- creează
- adaugă
- șterge

```
Subalgoritm creează (Epost, arb) este:
    creează (s)
    pentru fiecare e din Epost execută
        dacă e este operand atunci:
            alocă (nou)
            [nou].e ← e
            [nou].st ← NIL
            [nou].dr ← NIL
            adaugă (s, nou)
        altfel
            șterge(s, p1)
            șterge(s, p2)
            alocă (nou)
            [nou].e ← e
            [nou].st ← p2
            [nou].dr ← p1
            adaugă (s, nou)
        sf_dacă
    sf_pentru
    șterge(s, p)
    arb.răd ← p
sf_subalgoritm
```

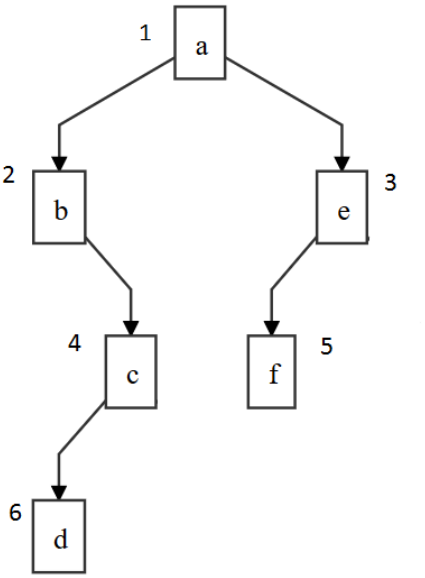
Dacă vrem să folosim interfața AB: Stiva va conține elemente de tip AB și folosesc operații din interfața AB

creeazăFrunză(ab, e)
adaugă(s, ab)

creeazăArbore(ab, p2, e, p1)
adaugă(s, ab)

șterge(s, arb)

2. Să se genereze tabelul corespunzător arborelui (se numerotează nodurile pe nivele)



	1	2	3
	Info	Index St	Index Dr
1	a	2	3
2	b	0	4
3	e	5	0
4	c	6	0
5	f	0	0
6	d	0	0

- Împărțim soluția în 2 funcții: Numerotare, Parcurgere
- Folosim o coadă pentru a reține nodurile (trebuie să parcurgem pe nivele)
- Presupunem că Nod are un câmp nr: Întreg (unde vom reține numărul nodului)

```
Subalgoritm numerotare (arb,k)
    k ← 0
    creează(c)
    dacă arb.răd ≠ NIL atunci
        adaugă(c, arb.răd)
        k ← 1
        [arb.răd].nr ← k
    sf_dacă
    câttimp (¬ vidă (c)) execută
        șterge (c, p)
        dacă ([p].st ≠ NIL) atunci
            k ← k + 1
            [[p].st].nr ← k
            adauga(c, [p].st)
        sf_dacă
        dacă ([p].dr ≠ NIL) atunci
            k ← k + 1
            [[p].dr].nr ← k
            adauga(c, [p].dr)
        sf_dacă
    sf_câttimp
sf_subalgoritm
```

```
subalgoritm parcurgere(p, T) este:
    dacă (p ≠ NIL) atunci
        T[[p].nr, 1] ← [p].e
        dacă ([p].st ≠ NIL) atunci
            T[[p].nr, 2] ← [[p].st].nr
        altfel
            T[[p].nr, 2] ← 0
        sf_dacă
        dacă ([p].dr ≠ NIL) atunci
            T[[p].nr, 3] ← [[p].dr].nr
        altfel
            T[[p].nr, 3] ← 0
        sf_dacă
        parcurgere([p].st, T)
        parcurgere([p].dr, T)
    sf_dacă
sf_subalgoritm

subalgoritm table(arb, T, k) este:
    numerotare(arb, k)
    @creem T cu k linii
    parcurgere(arb.răd, T)
sf_subalgoritm
```

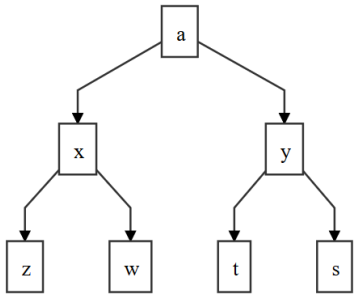
- Ce se întâmplă dacă nu vreau să am camp nr în Nod?
    - Dacă fac o singură funcție (numerotare + parcurgere împreună) atunci pot pune în coadă perechi <nod, nr>.
    - Dacă folosesc 2 funcții, funcția numerotare poate să creeze un Dictionar cu elemente <nod, nr> sau <arb, nr>, care să fie transmis funcției parcurgere.
- Ce se întâmplă dacă nu vreau ca parcurgere să fie recursiv?
    - Folosesc o coadă sau stivă pentru parcurgere

3. Să se creeze un arbore pe nivele. Se dau informațiile astfel:
   Rădăcina: 1
   Descendenții lui 1: 2, 5
   Descendenții lui 2: 0, 3
   Descendenții lui 5: 6, 0
   Descendenții lui 3: 4, 0
   Descendenții lui 6: 0, 0
   Descendenții lui 4: 0, 0

```
Funcția creeazăNod(e)
    @returnează un pointer la un Nod care conține e ca informație și cei doi
fii sunt NIL
Sf_functie
```

```
Subalgoritm creeazăNivele(arb)
    scrie ”Rădăcina:”
    citește e
    dacă e ≠ 0 atunci
        arb.răd ← creeazăNod(e)
        creează(c)
        adaugă(c, arb.răd)
        câttimp ¬ vidă(c) execută
            șterge(c, p)
            scrie ”Descendenții lui ” [p].e
            citește e1, e2
            dacă e1 ≠ 0 atunci
                p1 ← creeazăNod (e1)
                [p].st ← p1
                adaugă(c, p1)
            sf_dacă
            dacă e2 ≠ 0 atunci
                p2 ← creeazăNod (e2)
                [p].dr ← p2
                adaugă(c, p2)
            sf_dacă
        sf_câttimp
    altfel
       arb.răd ← NIL
    sf_dacă
sf_subalgoritm
```

4. Să se construiască un arbore care reprezintă strămoșii unei personae până la generația a ”n”-a, arborescența stângă reprezentând linia maternă, iar cea dreaptă linia paternă.



   a. Să se afișeze toate persoanele de sex feminine (presupunem că rădăcina e femeie)
      - a, x, z, t
   b. Să se afișeze toți strămoșii de gradul k (rădăcina are gradul 0)
      - k = 2 – z, w, t, s

a. Se parcurge arborele, folosind o coadă (sau stivă) și se tipăresc doar fii stângi

```
Subalgoritm femei (arb) este:
    creeaza(c)
    dacă arb.răd ≠ NIL atunci
        adaugă (c, arb.răd)
        scrie arb.răd
    sf_dacă
    câttimp ¬vidă(c) execută
        șterge(c, p)
        dacă ([p].st ≠ NIL) atunci
            scrie [[p].st].e
            adaugă(c, [p].st)
        sf_dacă
        dacă ([p].dr ≠ NIL) atunci
            adaugă(c, [p].dr)
        sf_dacă
    sf_câttimp
sf_subalgoritm
```

b. Varianta recursivă – folosim interfata arborelui (nu intrăm în detalii de reprezentare)

```
Subalgoritm nivel(arb, k, v) este
// v este un vector în care vom pune elementele de pe nivelul k
    dacă ¬vid(arb) atunci
        dacă k = 0 atunci
            adaugă(v, element(arb))
        altfel
            dacă ¬vid(stâng (arb)) atunci
                nivel(stâng (arb), k-1, v)
            sf_dacă
            dacă ¬vid(drept (arb)) atunci
                nivel(drept (arb), k-1, v)
            sf_dacă
        sf_dacă
    sf_dacă
sf_subalgoritm

Subalgoritm strămoși (arb, k, v) este:
    creeaza (v) // inițializăm un vector vid
    nivel (arb, k, v)
    pentru i ← 1, dim(v) execută
        scrie element(v, i)
    sf_pentru
sf_subalgoritm
```

- Cum rezolv cu o funcție nerecursivă?
  - Într-o coadă/stivă pun perechi <p, nivel>