SISTEM DE CALCUL IN TIMP REAL

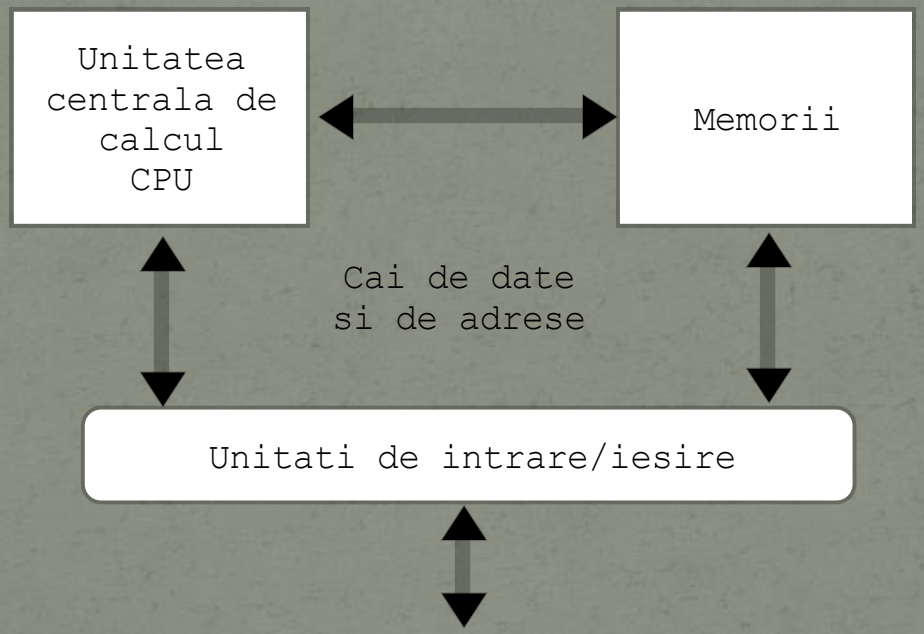
SCTR

-SZOKE ENIKO -

Curs 4 continuare curs 3

Cuprins

Arhitectura de baza a unui SC



Clase de arhitecturi ale unitatii centrale de calcul

In functie de complexitatea modului de adresare utilizat de un anumit tip de procesor:

- arhitecturi de 0-adrese
- arhitecturi de 1-adresa
- arhitecturi de 2-adrese
- arhitecturi de 3-adrese

Arhitecturi de 0-adrese sau arhitecturi de tip stiva

- CPU lucreaza numai cu un registru acumulator si o memorie de tip stiva ambele neadresabile.
- Instructiunile nu au in ele camp de adresa si SC nu are memorii adresabile.
- Modurile de adresare este cel implicit sau cel imediat.
- Se mai poate utiliza instructiuni de tip PUSH sau POP (stiva)

si operatii aritmetice sau logice (stiva)

Memorie de tip stiva - curs 3

Unele probleme care se rezolvă în mod eficient de calculatoarele orientate pe stivă sunt:

- transformarea expresilor aritmetice din forma infixată în forma postfixată
- evaluarea expresiilor aritmetice

Forma poloneza infixata

Forma poloneza postfixata

Forma poloneza prefixata

Forma poloneză **postfixată** numită şi **forma poloneză inversă,** este cea mai utilizată. Avantajele față de notația infixată sunt:

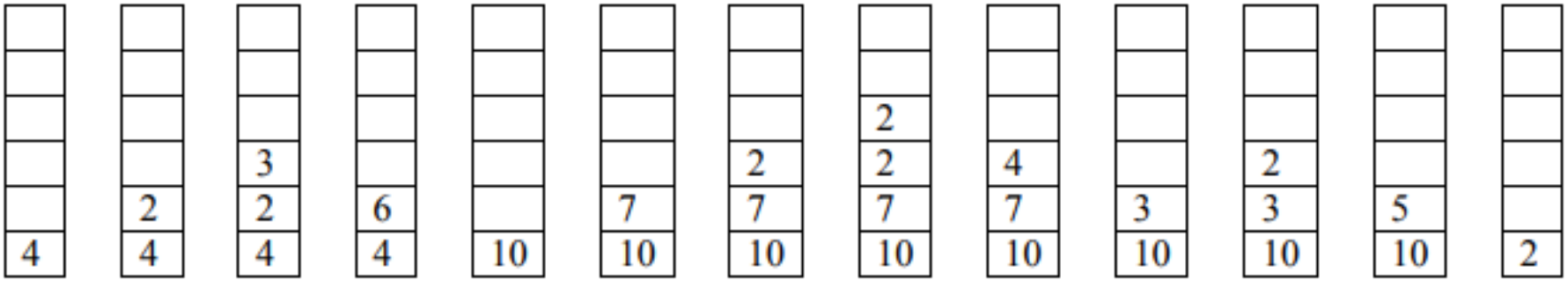
- Expresiile pot fi scrise fără paranteze,
- Regulile de precedență ale operatorilor infixați nu mai sunt necesare,
- Evaluarea expresiilor se realizează mai simplu la calculatoarele care dispun de memorie stivă.

Exemple:

Expresie infixată	Expresie postfixată
A*(B+C)	ABC+*
A*B+C	AB*C+
A*B+C*D	AB*CD*+
(A+B)*(C – D)	AB+CD – *
((A+B)*C – D) /(E–F)	AB+C*D – EF – /

Exemplu:
Expresie infixată: (4+2*3)/(7-2*2+2)
Expresie postfixată: 4 2 3 * + 7 2 2 * - 2 + /

Evaluare folosind stiva:



Utilizarea stivelor la apeluri de proceduri

Stiva reprezintă un dispozitiv frecvent utilizat într-un calculator. Domeniile principale ale stivelor sunt:

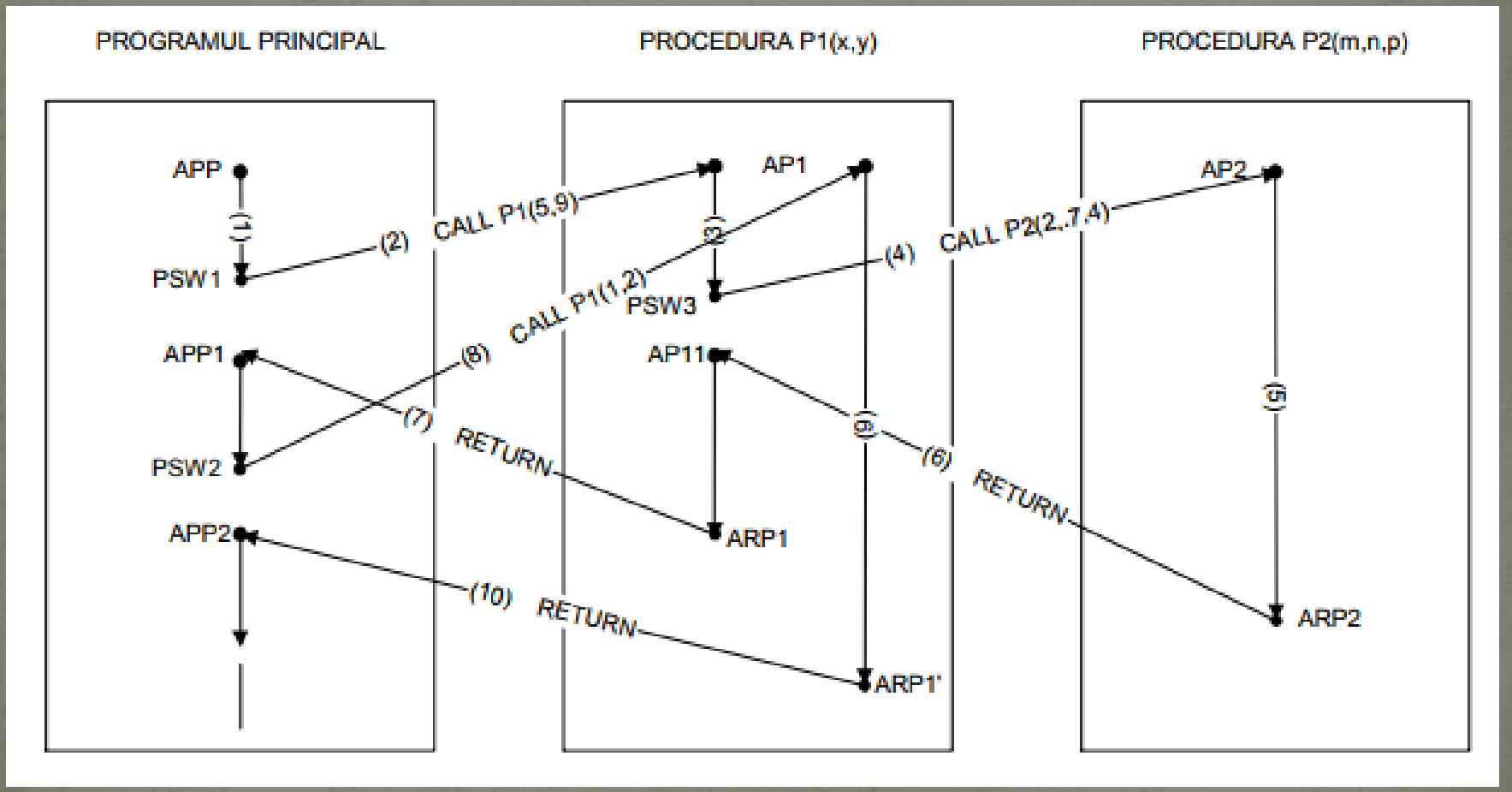
1) În calculatoarele tip stivă (stack computers) asigură efectuarea operațiilor aritmetice utilizând notația poloneză inversă (postfix).
2) La tratarea întreruperilor. Pentru deservirea unei cereri de întrerupere se opreşte execuția programului principal şi se sare la programul de servire al întreruperii. Înaintea saltului trebuie salvat contextul programului principal, deci al programului care se întrerupe. În principal trebuie salvată adresa de revenire în programul principal, după terminarea execuției rutinei de serviciu. Această salvare a contextului se face cu ajutorul stivei.
3) La apelul de proceduri. Dacă în timpul execuției unui program se apelează o procedură (subrutină), atunci se întrerupe execuția programului principal şi se sare la începutul procedurii apelate. Apelul se realizează printr-o procedură de tip CALL.

Procedura se caracterizează prin propriile variabile numite variabile locale. La scrierea inițială a procedurii se folosesc variabile fictive dar în momentul apelului procedurii, acestor variabile trebuie să li se atribuie valori concrete.

Există diferite variante de atribuire dar mai simplu este specificarea efectivă a valorii variabilelor locale direct în instrucțiunea CALL. Trebuie să se respecte riguros ordinea de definire a variabilelor din procedură precum şi tipul datelor.

După lansarea în execuție a procedurii, aceasta se execută în mod normal până la întâlnirea instrucțiunii RETURN. Atunci se revine în programul principal. Uneori se poate apela din interiorul procedurii o altă procedură (principiul procedurilor încuibate).

Exemplu: Fie un program principal PP şi două proceduri P1 şi P2. Procedura P1 este defintă cu 2 variabile fictive. Procedura P2 este definită cu 3 variabile fictive. Se presupune următoarea evoluție a programului descrisă în figura de mai jos.



Unde:

APP – adresa de start a programului principal

AP1 – adresa de start a procedurii P1

AP2 – adresa de start a procedurii P2

PSW1 – cuvântul de stare al programului principal ce trebuie salvat pe stivă atunci când se face salt la procedura P1

APP1 – adresa de retur după apelul procedurii P1

PSW3 – cuvântul de stare al procedurii P1 ce trebuie salvat pe stivă atunci când se face salt la procedura P2

AP11 – adresa de retur după apelul procedurii P2 din P1

ARP2 – adresa unde se apelează RETURN în procedura P2

ARP1 – adresa unde se apelează prima dată RETURN în procedura P1

PSW2 – cuvântul de stare al programului principal ce trebuie salvat pe stivă atunci când se face salt a doua dată la procedura P1

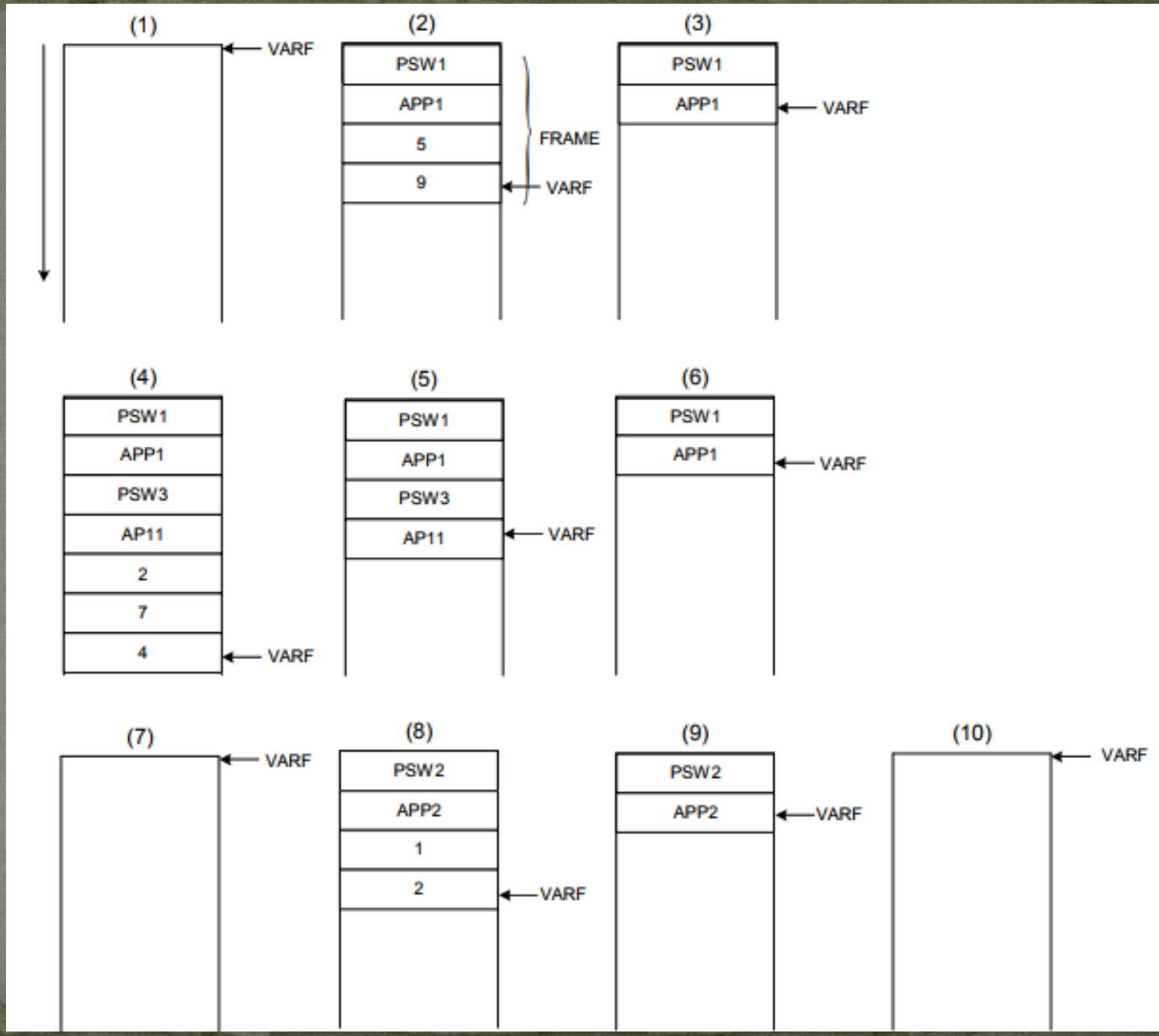
APP2 – adresa de retur după apelul procedurii P1 pentru a doua dată

ARP1' – adresa unde se apelează a doua dată RETURN în procedura P1

Procedura P1 are două variabile notate x şi y. Când se apelează procedura P1 se plasează în stivă, în cadrul frame-ului, şi cele două valori efective pentru cele două variabile. Acestea se depun în partea superioară a stivei, deasupra adresei de revenire şi a cuvântului de stare, deoarece primele operaţii la execuţia procedurii, presupune operaţii POP pentru preluarea din vârful stivei a valorilor efective pentru variabilele locale.

Aşa cum s-a arătat mai sus în exemplul dat, în procedura P1 apare un apel către procedura P2 care are 3 variabile m, n, p. La terminarea execuţiei procedurii P2 se execută instrucţiunea RETURN şi se revine la procedula apelantă P1 la adresa AP11. Se reia execuţia procedurii P1 până la terminarea ei adică la întâlnirea unei instrucţiuni RETURN, unde are loc revenirea în programul principal. Se continuă execuţia programului principal şi apare din nou o cerere de procedură P1 în alt context. Se execută procedura P1 care pentru datele noi nu mai întâlneşte un apel al procedurii P2. Apoi revine în programul principal şi se continuă execuţia programului până la ultima instrucţiune.

Evoluţia stivei pentru exemplul de mai sus este prezentată în figura următoare, unde la fiecare CALL se depune un frame în stivă, iar la fiecare RETURN se extrage un frame.


(1)
VARF
(2)
PSW1
APP1
5
9
FRAME
VARF
(3)
PSW1
APP1
VARF
(4)
PSW1
APP1
PSW3
AP11
2
7
4
VARF
(5)
PSW1
APP1
PSW3
AP11
VARF
(6)
PSW1
APP1
VARF
(7)
VARF
(8)
PSW2
APP2
1
2
VARF
(9)
PSW2
APP2
VARF
(10)
VARF

Arbore binar

Arborii sunt folosiți în general pentru a modela o **ierarhie de elemente**

În funcție de elementele ce pot fi reprezentate în noduri şi de restricțiile aplicate arborelui, se pot crea structuri de date cu proprietăți deosebite: arbori binari de cautare, heap-uri, arbori AVL, arbori roşu-negru şi multe altele.

Arbore binar

Un **arbore binar** reprezinta o multime recursiva de N noduri. Fiecare nod poate avea cel mult doua subramuri care leaga acel nod de noduri de pe un alt nivel. Unul dintre cei N noduri este **nodul radacina** iar nodurile fara nici o subramura se numesc **noduri terminale**.

Pentru a programa un procesor cu arhitectura de 0-adrese sa calculeze o expresie aritmetica sau logica, se utilizeaza un arbore binar in a carui nodurile terminale se aseaza operanzii expresiei iar in celelalte noduri se aseaza operatorii algebrici sau logici.

sortare, cautare, parcurgere in inordine,

Arborele binar asociat unei expresii aritmetice

O expresie aritmetică (matematică) este un şir de caractere compus din:

- variabile
- constante
- operatori
- (eventual) paranteze.

Fiecărei expresii aritmetice i se poate asocia un arbore binar, numit arborele sintactic, în care:

- nodurile interioare reprezintă **operatorii: +, -, *, /, %** ;
- frunzele reprezintă **constantele** sau **variabilele**.

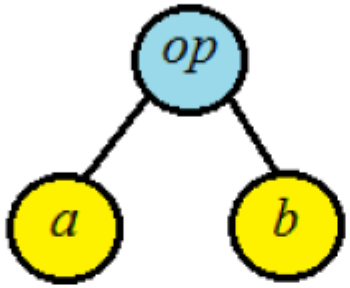
Arborele sintactic este construit după următoarele reguli:

A1) dacă expresia este formată dintr-un singur operand, arborele binar asociat are un singur nod, ***nod rădăcină***, care conţine drept informaţie utilă operandul respectiv. **De exemplu**, sunt reprezentaţi operanzii *a*, 12:

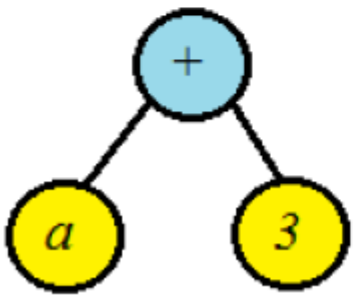
E = *operand* $\rightarrow$ 

A2) dacă expresia este de forma E = *a* op *b*, unde *op* este un operator binar iar *a* şi *b* sunt doi operanzi, i se asociază un arbore binar care are nodul rădăcină etichetat cu operatorul respectiv, subarborele stâng este operandul a iar subarborele drept este operandul b.

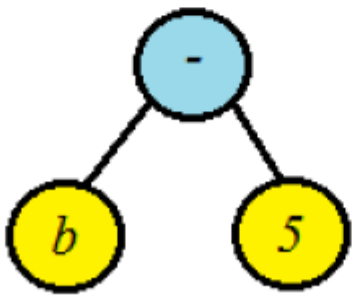
E = *a* op *b* →



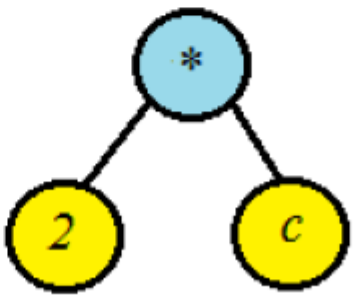
De exemplu, arborii asociaţi expresiilor E_1 = a+3, E_2 =b-5, E_3 =2*c, E_4 = d/4 sunt următorii:



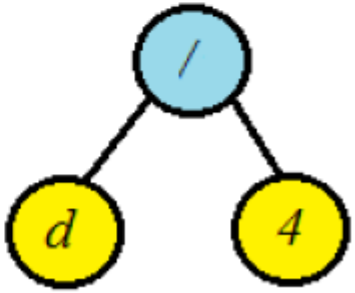
E1=a+3



E2=b-5



*E3=2*c*

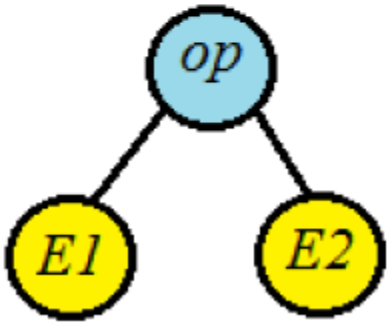


E4=d/4

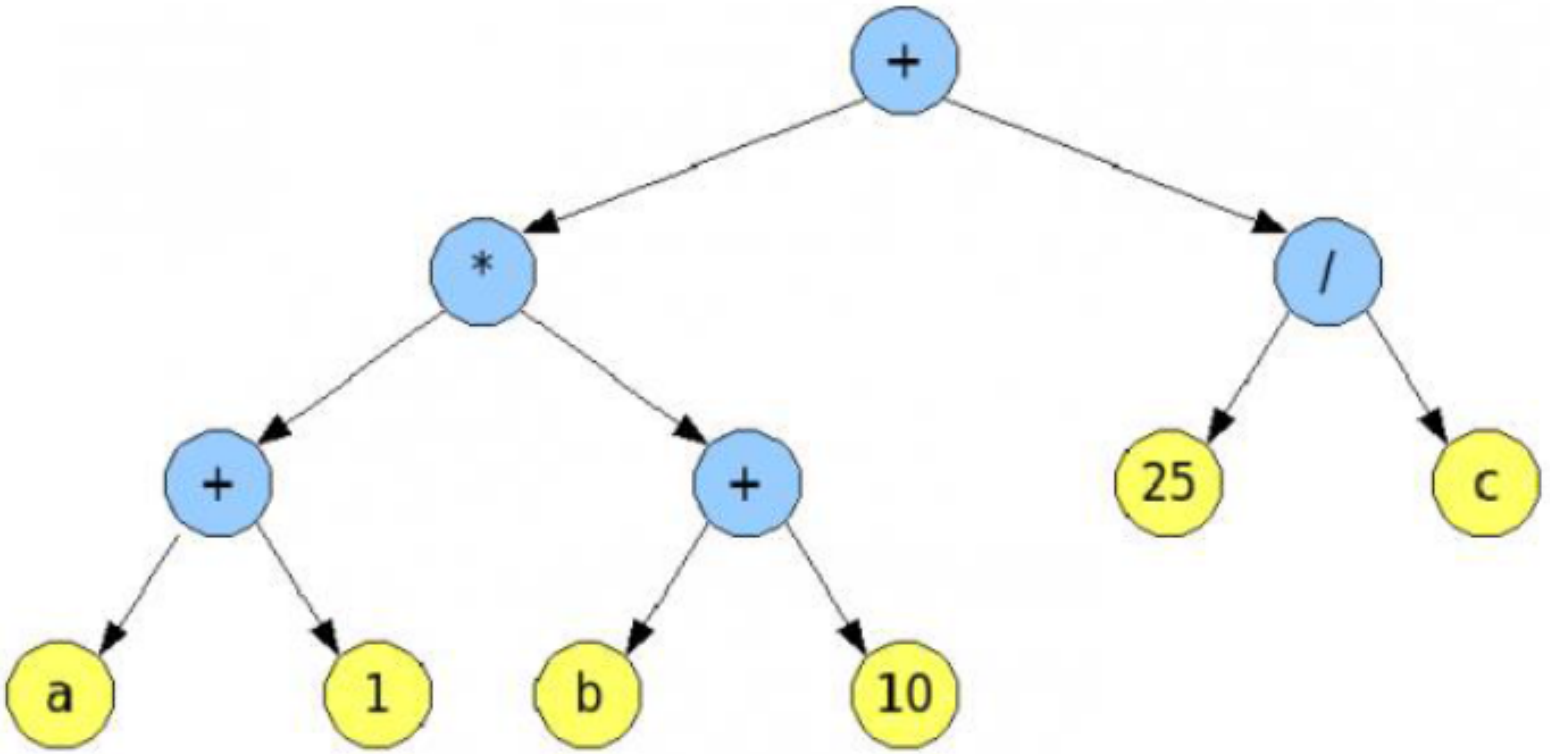
A3) dacă E=(E1), unde E1 este o expresie aritmetică, atunci arborele binar asociat lui E coincide cu arborele binar asociat lui E1.

A4) dacă expresia este de forma E = E_1 op E_2, unde *op* este un operator binar iar E_1 şi E_2 sunt alte expresii, i se asociază un arbore binar care are nodul rădăcină etichetat cu operatorul respectiv, subarborele stâng arborele binar asociat expresiei E_1 şi subarborele drept arborele binar asociat expresiei E_2 :

E = E1 op *E2* →



Exemplu: Pentru expresia ***(a+1)*(b+10)+25/c*** , arborele asociat este prezentat mai jos:



Observaţii:

AO1) În arborele binar asociat unei expresii **nu apar parantezele**.

AO2) ***Pentru operaţiile comutative*** **+, *** ordinea subarborilor stâng şi drept nu contează, aceştia pot fi inversaţi iar arborele este bine construit. În cazul operaţiilor necomutative **-**, /, % ordinea subarborilor este strict precizată, adică subarborii nu pot fi inversaţi în reprezentare.

AO3) Arborele binar asociat unei expresii ***nu este unic***, în general. Se pot obţine arbori sintactici diferiţi dacă se interschimbă subarborii în cazul operaţiilor comutative, iar pentru unele expresii există mai multe posibilităţi de alegere a rădăcinii arborelui sau a subarborilor.

Forma poloneză asociată unei expresii

Notaţia poloneză a expresiilor aritmetice reprezintă una dintre cele mai importante aplicaţii ale arborilor binari. Această notaţie a fost introdusă de matematicianul polonez **J. Lukasiewicz** în 1920. Forma poloneză a expresiilor aritmetice este utilă în faza de compilare a programelor ca formă intermediară pentru generarea codului obiect, necesar în faza de execuţie pentru evaluarea acelei expresii.

Forma poloneză prefixată

Forma poloneză prefixată a unei expresii aritmetice E, sau ***notaţia prefixată*** asociată expresiei E, se obţine prin parcurgerea în preordine a unui arbore binar asociat expresiei E.

Forma poloneză prefixată, notată cu ***fpp***, poate fi obţinută şi dintr-o expresie aritmetică direct, prin aplicarea următoarelor reguli:

FPP1) Dacă expresia este formată dintr-un singur operand, *fpp* coincide cu acel operand.

FPP2) Pentru o expresie de forma E=*a op b*, avem *fpp*(E) = *op a b*.

De exemplu, pentru expresiile *a+b*, *a-b*, *a*b*, *a / b*, *c* % b, avem:

fpp(a+b) = + a b, fpp(a-b) = - a b, fpp(a*b) = * a b, fpp(a/b) = / a b, fpp(a%b) = % a b.

FPP3) Pentru o expresie cu paranteze de forma E=*(E1)*, *fpp asociată lui* E coincide cu *fpp asociată expresiei E1*, *fpp(E)=fpp(E1)*.

FPP4) Pentru o expresie de forma E=*E1 op E2*, FPP= *op fpp(E1) fpp(E2)* .

Forma poloneză postfixată

Forma poloneză postfixată a unei expresii aritmetice E, sau ***notaţia postfixată*** asociată expresiei E, se obţine prin parcurgerea în postordine a unui arbore binar asociat expresiei E.

Forma poloneză postfixată, notată cu ***fpt***, poate fi obţinută şi dintr-o expresie aritmetică direct, prin aplicarea următoarelor reguli:

FPT1) Dacă expresia este formată dintr-un singur operand, *fppt* coincide cu acel operand.

FPT2) Pentru o expresie de forma **E=*a op b***, avem ***fpt*(E) = *a b op*.**

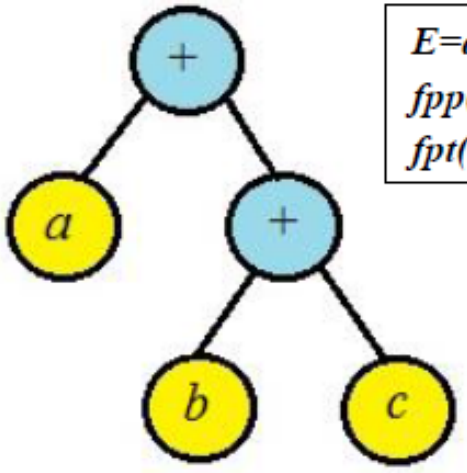
De exemplu, pentru expresiile ***a+b, a-b, a*b, a / b, c % b***, avem:

fpt(a+b) = a b +, fpt(a-b) = a b -, fpt(a*b) = a b *, fpt(a/b) = a b /, fpt(a%b) = a b %.

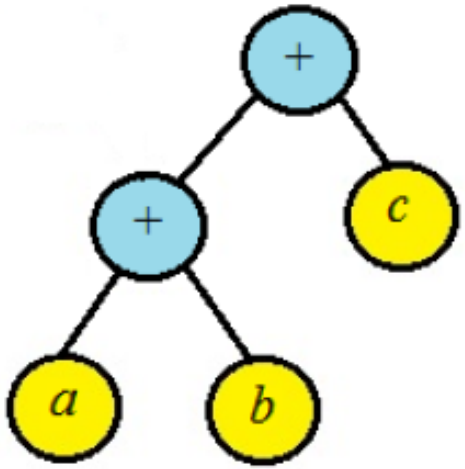
FPT3) Pentru o expresie cu paranteze de forma **E=*(E1)***, *fpt asociată lui* E coincide cu *fpt asociată expresiei E1*, ***fpt(E)=fpt(E1)***.

FPT4) Pentru o expresie de forma E=*E1 op E2*, FPT= *fpt(E1) fpt(E2) op*.

Exemple: 1) Expresiei aritmetice **E=*a+b+c*** îi pot fi asociaţi, de exemplu, arborii binari:

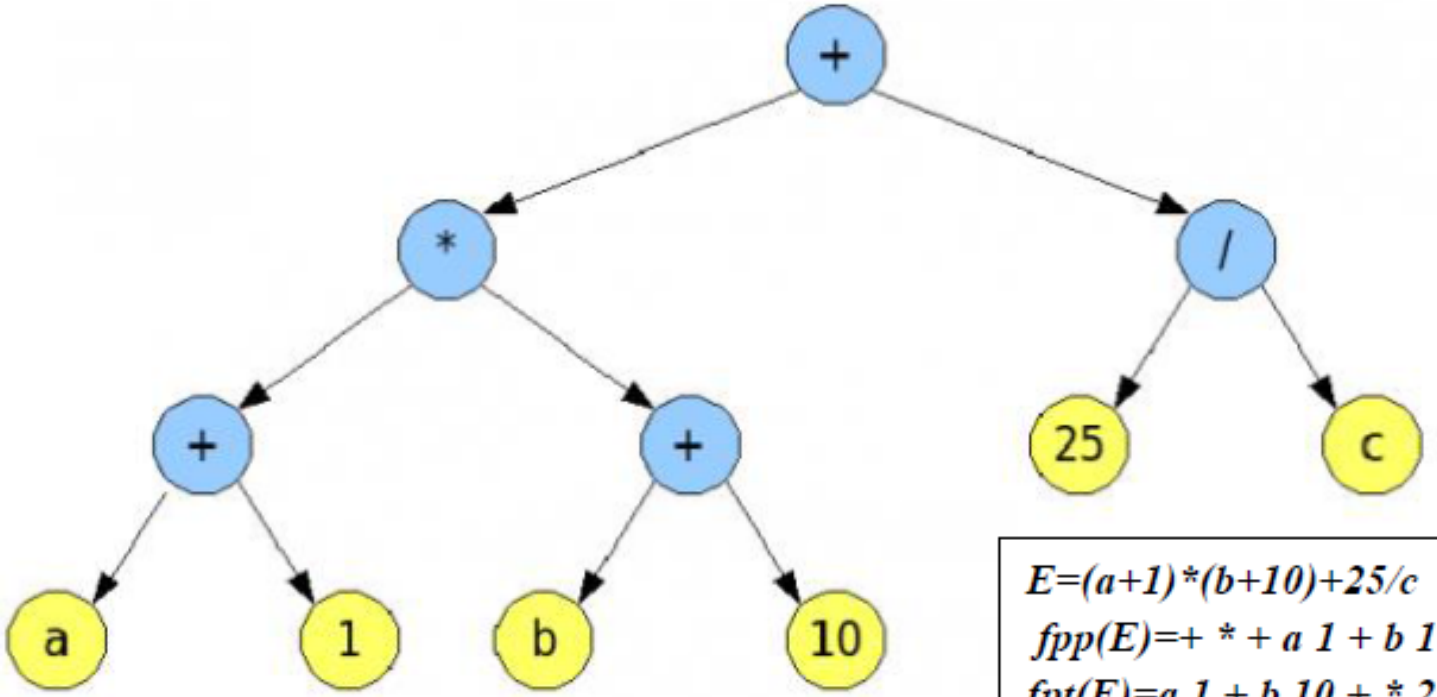


E=a+b+c
fpp(E)=+ a + b c
fpt(E)=a b c + +



E=a+b+c
fpp(E)=+ + a b c
fpt(E)=a b + c +

2) Pentru expresia ***(a+1)*(b+10)+25/c*** , cu arborele sintactic asociat alăturat, *formele poloneze prefixate* şi *postfixate* sunt:



E=(a+1)*(b+10)+25/c
fpp(E)=+ * + a 1 + b 10 / 25 c
fpt(E)=a 1 + b 10 + * 25 c / +

Observaţii:

FPO1) *Forma poloneză prefixată (postfixată)* a unei expresii ***nu conţine paranteze***.

FPO2) *Forma poloneză prefixată (postfixată)* a unei expresii ***nu este unică***.

FPO3) ***Operanzii apar scrişi în aceeaşi ordine*** ca şi în notaţia infixată a expresiei aritmetice.

Aplicaţii:

1. Reprezentaţi arborii binari asociaţi expresiilor aritmetice alăturate şi precizaţi *forma poloneză prefixată* şi *postfixată* a expresiei:

Expresia	*Arborele sintactic asociat*	*Forma poloneză*
E1 = *a*b+c/d–e*		
E2 = *a*(b+c)/(d–e)*		
E3 = *(a+b)*(c–d%e)*		
E4 = *(a+b)*(b+c)*(c+a)*		
E5 = $\frac{a}{a+b} + \frac{a+b}{b}$		
E6 = $\frac{a+b}{a-b} + \frac{a-b}{a+b}$		

$E7 = \frac{x}{y+z} + \frac{y}{x+z} + \frac{z}{x+y}$		
$E8 = \frac{x}{x+y-z} + \frac{y}{x-y+z} + \frac{z}{z+y-x}$		

2. Deduceţi scrierea în forma obşnuită (*notaţia infixată*) a următoarelor expresii aritmetice, date în *forma poloneză prefixată*, ştiind că toate constantele care apar sunt formate dintr-o singură literă sau cifră şi desenaţi arborele sintactic:

a) + * *a b* / *a b*

b) / * *a* + *b c d*

c) * *a* / + *b c d*

d) / + *a b* – *a b*

e) / * + *a b c* % – *x y z*

f) / *x* + *y* / % *x* 2 *z*

g) – * 2 + *a b* / + *a* 1 *b*

h) / * + *a* 1 – *b* 2 % + *a b c*

i) / * + *a b c* % – *x y z*

Clasificarea arborilor binari

- arbori binari stricți;
- arbori binari plini;
- arbori binari compleți;
- arbori binari degenerați;
- arbori binari echilibrați.

1. Arbori binari stricţi sunt arborii binari în care orice vârf are gradul zero (este terminal) sau doi (are exact doi fii).

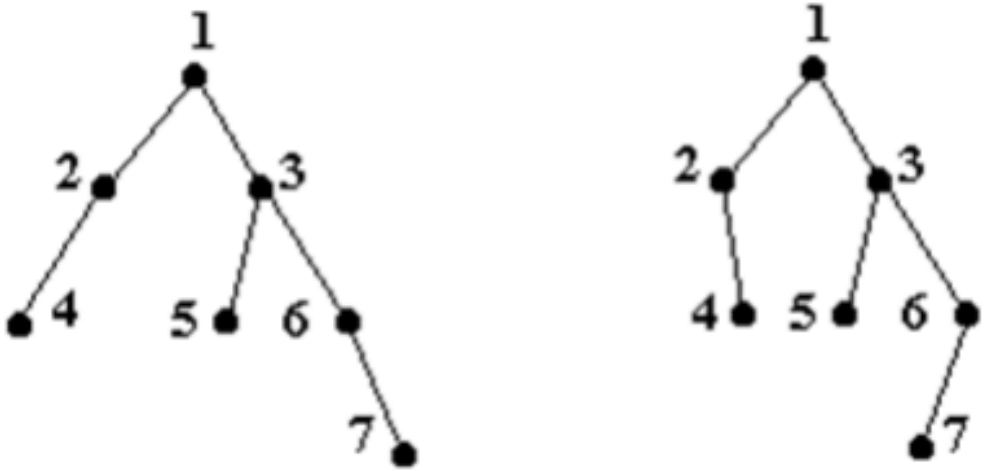


Figura nr. 1 – Arbori binari oarecare

De exemplu, arborii din Figura nr. 1 nu sunt arbori binari stricţi (nodurile 2 şi 6 având un singur fiu), dar în Figura nr. 2 este un arbore binar strict.

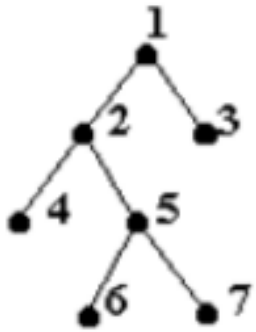


Figura nr. 2 – Arbore binar strict

2. *Arbori binari plini* sunt arbori binari care au 2^k-1 vârfuri dispuse pe nivelurile 0, 1, ... , k-1, astfel încât pe fiecare nivel i se găsesc 2^i vârfuri.

De exemplu, arborele binar plin cu înălţimea 2 se prezintă astfel:

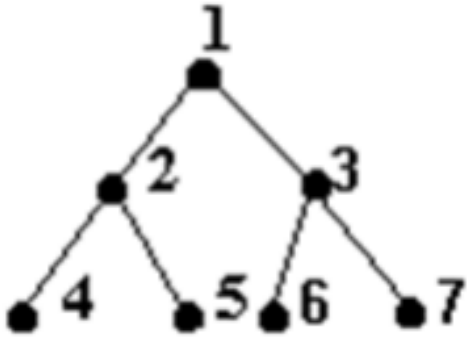


Figura nr. 3 – Arbore binar plin

3. Arborii binari compleţi sunt arbori binari care se obţin dintr-un arbore binar plin prin eliminarea din dreapta către stânga a unor noduri de pe ultimul nivel. Mai exact, pentru a construi un arbore binar complet cu n noduri, determinăm k astfel încât

$2^k \leq n < 2^{k+1} \Leftrightarrow k = [\log_2 n]$.

Construim arborele binar plin cu $2^{k+1}-1$ noduri şi eliminăm de pe ultimul nivel nodurile $2^{k+1}-1$, $2^{k+1}-2$, ..., n+1.

De exemplu, arborele binar complet cu 5 vârfuri (Figura nr. 4) se obţine prin eliminarea vârfurilor 7 şi 6 din arborele binar plin cu înălţimea 2 (Figura nr.3):

Figura nr. 4 – Arbore binar complet

4. *Arbori binari echilibraţi* - sunt arbori binari în care, pentru orice nod,diferenţa dintre înălţimea subarborelui stâng şi înălţimea subarborelui drept este cel mult o unitate.
Înălţimea unui arbore este lungimea celui mai lung drum de la nodul rădăcină la unul din nodurile terminale (frunze).

De exemplu:

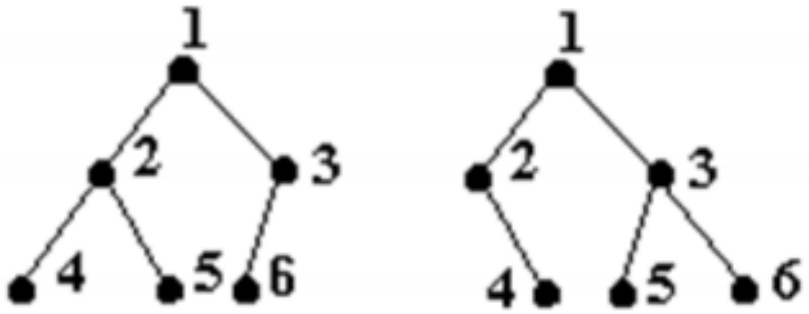


Figura nr. 5 - Arbori binari echilibraţi

5. *Arbori binari degeneraţi* - sunt arbori binari cu n vârfuri dispuse pe n niveluri.

De exemplu:

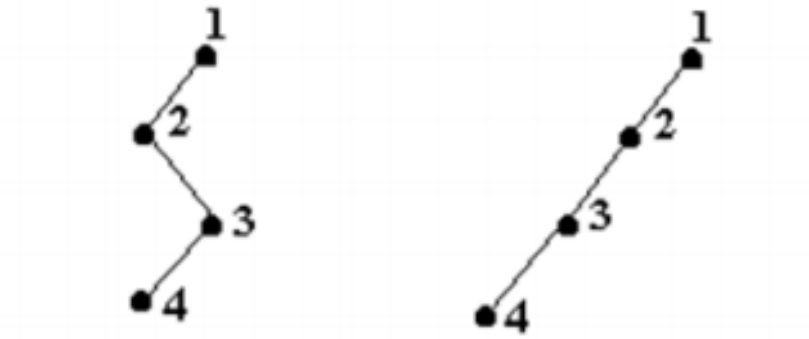
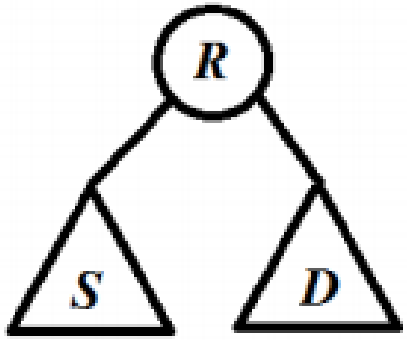


Figura nr. 6 – Arbori binari degeneraţi

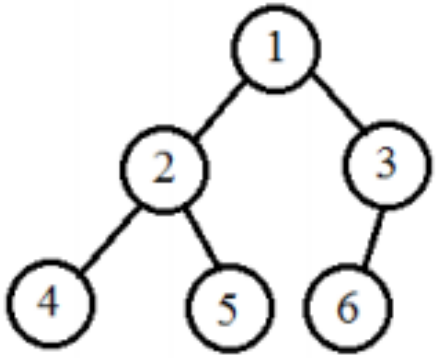
Parcurgerile arborilor



Se face întotdeauna o distincţie clară între cei doi subarbori. Dacă subarborele stâng S este nevid, rădăcina lui se numeşte fiul stâng al rădăcinii R. Analog, dacă subarborele drept D este nevid, rădăcina lui este fiul drept al rădăcinii R.

Parcurgerile arborilor binari sunt cele mai frecvente operaţii utilizate pe arbori. *Parcurgerea* unui arbore înseamnă vizitarea fiecărui nod al arborelui o singură dată, cu scopul prelucrării informaţiei memorate în acel nod. Dintre cele mai utilizate parcurgeri sunt parcurgerile în adâncime (*algoritmul* **DFS**) şi pe niveluri (*algoritmul* **BFS**).

A. Parcurgeri în adâncime sunt parcurgerile în preordine, inordine şi postordine. În toate cele trei tipuri de parcurgere în adâncime se vizitează prima dată subarborele stâng şi apoi subarborele drept, iar diferenţa constă în momentul în care se vizitează rădăcina. Funcţiile de parcurgere pot fi scrise recursiv sau iterativ, folosind o stivă.



n	1	2	3	4	5	6
S	2	4	6	0	0	0
D	3	5	0	0	0	0

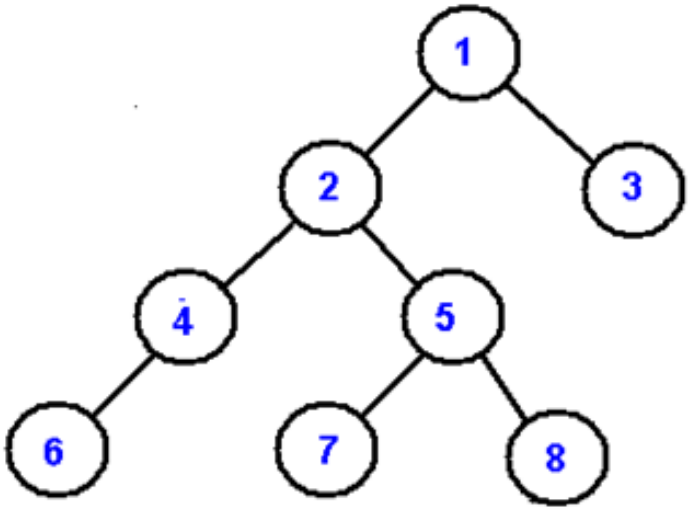
Parcurgerea în preordine RSD – se vizitează mai întâi rădăcina ***R***, apoi se parcurge în preordine subarborele stâng ***S*** şi subarborele drept ***D***. *Pentru arborele din figură, şirul parcurgerii* ***RSD*** *este:* **1 2 4 5 3 6**.

Parcurgerea în inordine SRD – se parcurge mai întâi în inordine suarborele stâng ***S***, apoi rădăcina ***R***, apoi se parcurge în inordine subarborele drept ***D***. *Pentru arborele din figură, şirul parcurgerii* ***SRD*** *este:* **4 2 5 1 6 3**.

Parcurgerea în postordine SDR – se parcurg în postordine subarborele stâng ***S*** şi subarborele drept ***D***, iar apoi se vizitează rădăcina ***R***. *Pentru arborele din figură, şirul parcurgerii* ***SDR*** *este:* **4 5 2 6 3 1**.

B. Parcurgerea pe niveluri – se vizitează rădăcina, apoi toţi fii nodului rădăcină, de la stânga spre dreapta şi se continuă în acest mod pe toate nivelurile. *Pentru arborele din figură, şirul parcurgerii pe niveluri este:* **1 2 3 4 5 6**.

Arbore binar



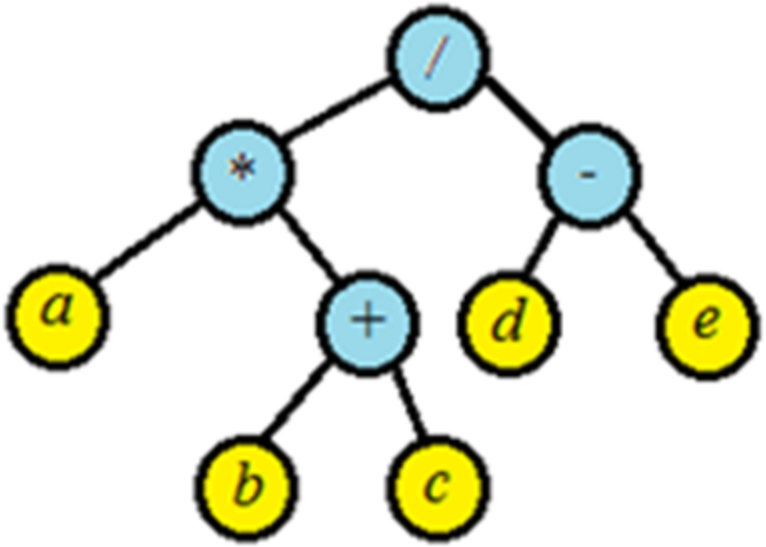
- *Reprezentarea arborelui:*

i:	1	2	3	4	5	6	7	8

- *h=3, niveluri=4*

RSD:
SRD:
SDR:

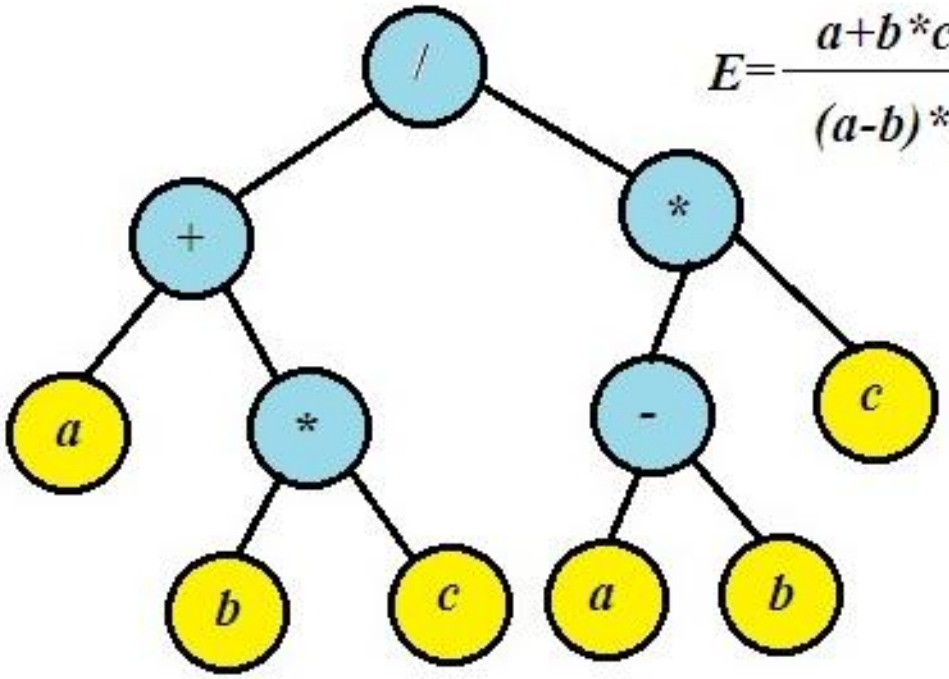
Forma poloneză



Expresia aritmetică:
E= *a*(b+c)/(d–e)*

forma poloneză prefixată:
***fpp*= / * a + b c - d e**

forma poloneză postfixată:
***fpt*=a b c + * d e - /**



$$E=\frac{a+b*c}{(a-b)*c}$$

cu albastru - operatorii cu galben - operanzii

Forma poloneză prefixată: / + a * b c * - a b c

Forma poloneză postfixată: a b c * + a b - c * /

Implementarea arborilor binari

În reprezentarea cu pointeri, un nod al arborelui binar este precizat printr-o structură alocată dinamic, care conține o cheie şi doi ponteri la nodurile succesori (şi eventual un pointer la nodul predecesor).

```
struct arb{
     void        *key;
     struct arb *stg;
     struct arb *dr;
     struct arb *pred;
};
```

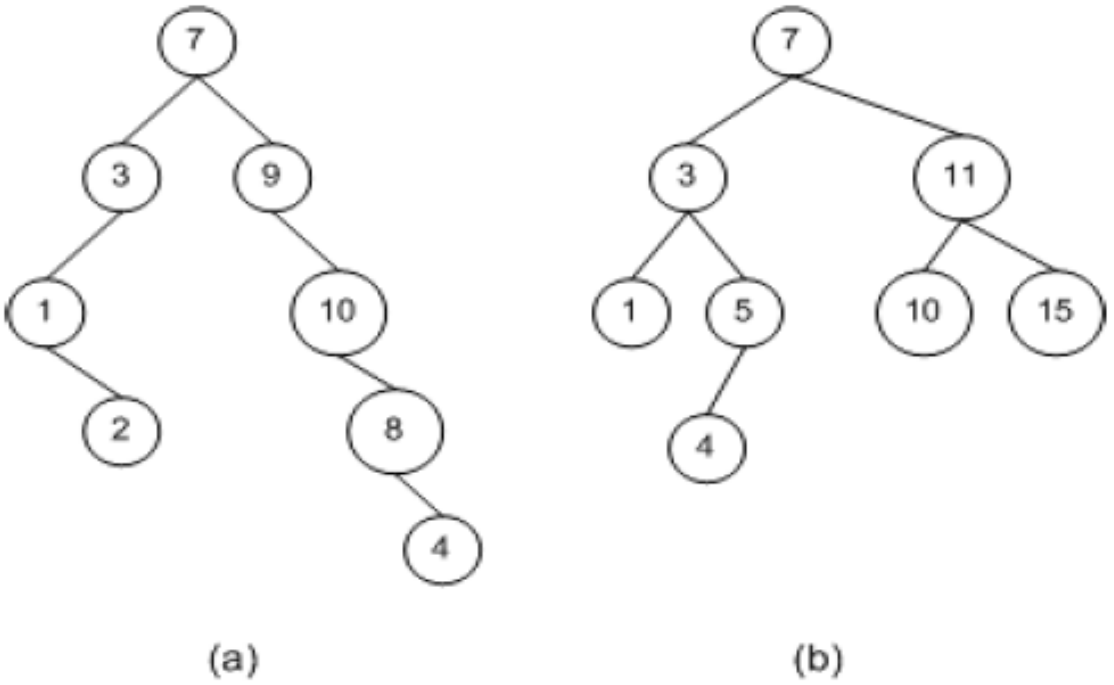
Arborele binar va fi precizat printr-un pointer la nodul rădăcină.

```
typedef struct arb *Arb;
```

Arbori Binari de Căutare

Un arbore binar de căutare este un arbore binar destinat îmbunătățirii timpului de cutarea informației. El va trebui să respecte următoarea proprietate: pentru oricare nod *n*, fiecare din descendentii din subarborele din stânga va avea valoarea informației mai mică decât a nodului *n*, iar fiecare din descendenții din subarborele din dreapta va avea valoarea informației mai mare decât a nodului n.

Exemplu:



Arbor binar de cautare ABC?
a) ,b)

a) Inserare = adaugarea unui nod într-un arbore de căutare,
cu păstrarea proprietăţii de arbore de căutare

Paşii pentru inserarea unui nod în arbore sunt:
Dacă arborele este nevid atunci avem cazurile:
i. valoarea coincide cu cheia vârfului - se renunţă la inserare,
deoarece cheile din arbore sunt **unice**;
ii. valoarea căutată este mai **mică** decât cheia asociată nodului,
căutarea continuă pentru subarborele **stâng**
iii. valoarea căutată este mai **mare** decât cheia asociată nodului,
căutarea continuă pentru subarborele **drept**

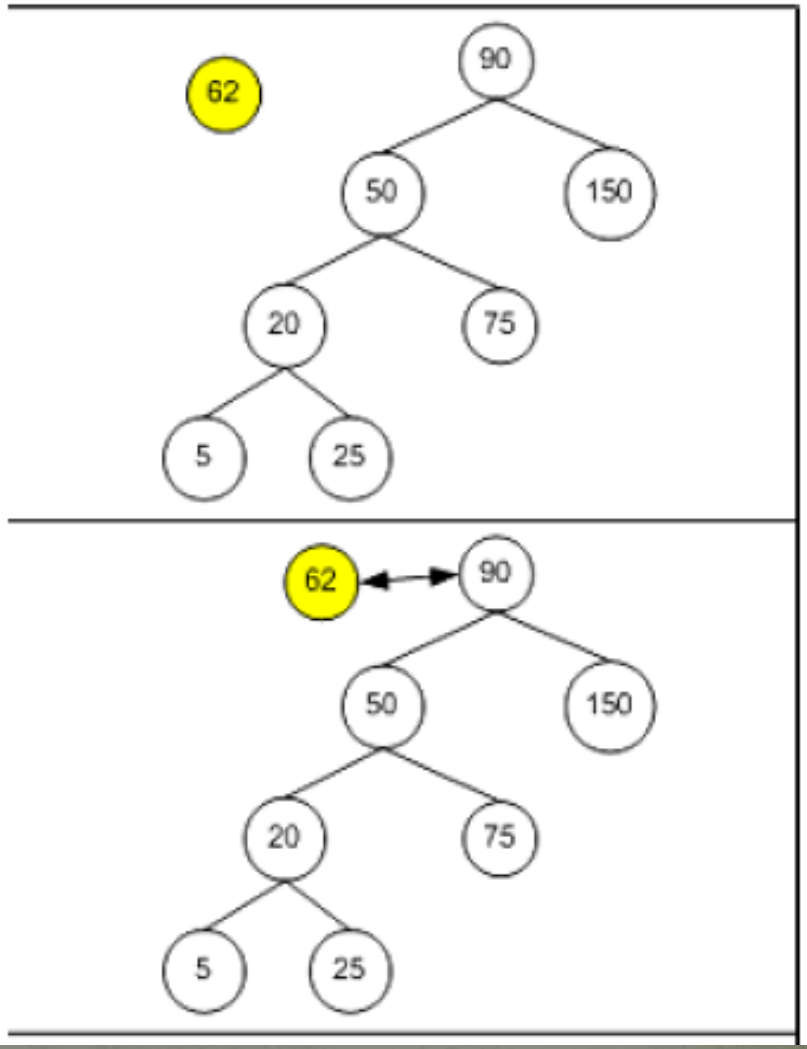
b) Căutarea unei valori în arbore.

Paşii pentru operaţia de căutare sunt:
Dacă arborele este nevid atunci avem cazurile:
i. valoarea coincide cu cheia vârfului - căutarea se opreşte cu succes,
am găsit valoarea în arbore;
ii. valoarea căutată este mai **mică** decât cheia asociată nodului,
căutarea continuă pentru subarborele **stâng**
iii. valoarea căutată este mai **mare** decât cheia asociată nodului,
căutarea continuă pentru subarborele **drept**

Crearea şi inserarea într-un arbore binar de căutare

Vom considera arborele din dreapta. Să presupunem că dorim să inserăm nodul cu valoarea 62. Acesta se va insera ca nod frunză.

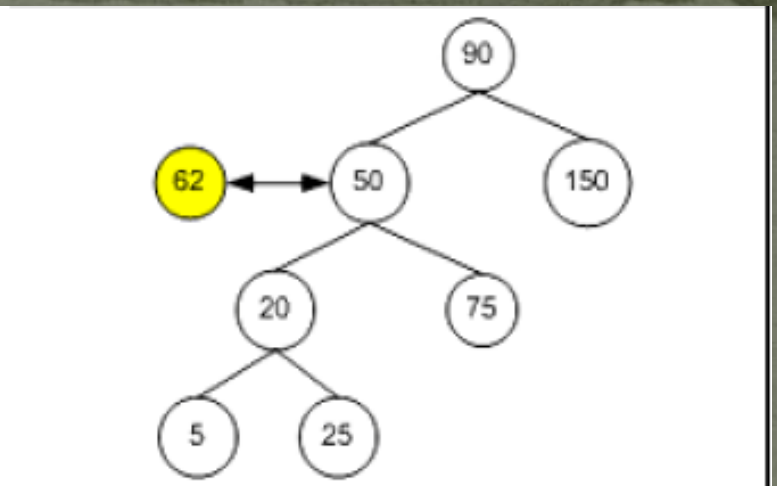
Pentru a-l insera va trebui să căutăm o poziție în arbore care respectă regula de integritatea a arborilor binari de căutare.



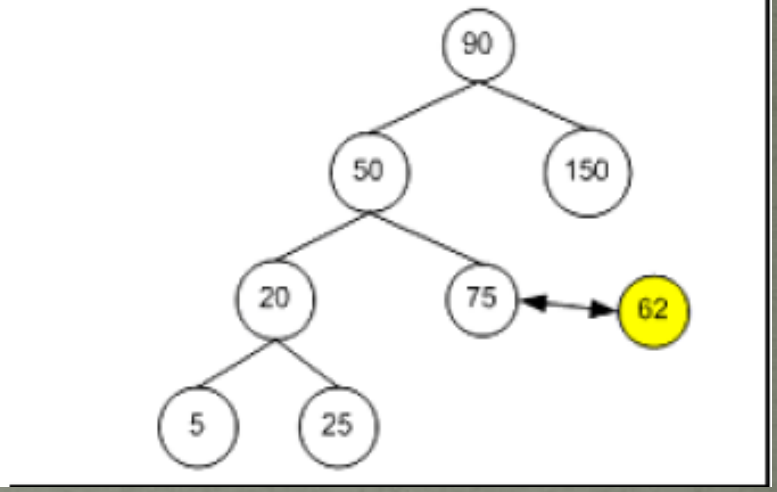
Vom începe prin compararea nodului de inserat (62), cu rădăcina arborelui (90).

Observăm că este mai mic decât ea, deci va trebui inserat undeva în subarborele stâng al acesteia.

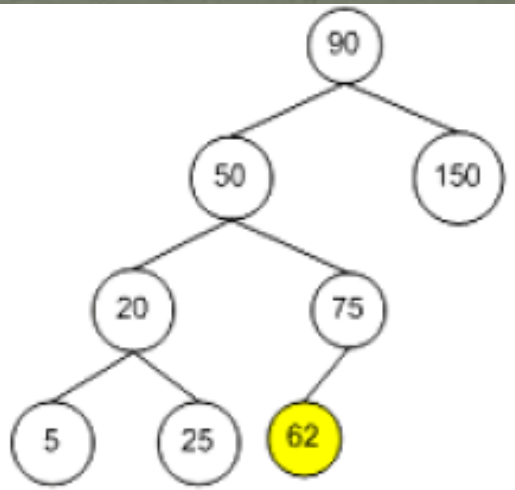
Vom compara apoi 62 cu 50. Din moment ce 62 este mai mare decât 50, nodul 62 va trebui plasat undeva în subarborele drept al lui 50.



Se compară apoi 62 cu 75. Deoarece 75 este mai mare decât 62, 62 trebuie să se afle în subarborele din stânga al nodului 75



Dar 75 nu are nici un copil în partea stângă. Asta înseamnă că am găsit locația pentru nodul 62. Tot ceea ce mai trebuie făcut este să modificăm în nodul 75 adresa către copilul din stânga, încât să indice spre 62.



c) Ştergerea unui nod din arbore

Paşii pentru operaţia de căutare sunt:

Dacă arborele este nevid atunci avem cazurile:

i. valoarea coincide cu cheia vârfului - am găsit valoarea în arbore

1. dacă nodul este **terminal** (subarborii stâng şi drept sunt vizi),
acesta este şters, iar adresa reţinută de nodul părinte pentru el va fi nulă;
2. dacă numai subarborele **drept** este nevid, nodul este şters,
iar nodul părinte va reţine, în locul adresei lui, adresa subarborelui drept;
3. dacă numai subarborele **stâng** este nevid, nodul este şters,
iar nodul părinte va reţine, în locul adresei lui, adresa subarborelui stâng;
4. dacă **ambii** subarbori sunt **nevizi**:
- se identifică cel mai din dreapta nod al subarborelui stâng;
- cheia nodului astfel identificat va fi memorată de nodul analizat;
- nodul astfel identificat se şterge, iar ştergerea se efectuează ca în cazul în care nodul subordonează numai subarborele stâng (3)

ii. valoarea căutată este mai **mică** decât cheia asociată nodului,
operaţia de ştergere continuă pentru subarborele **stâng**

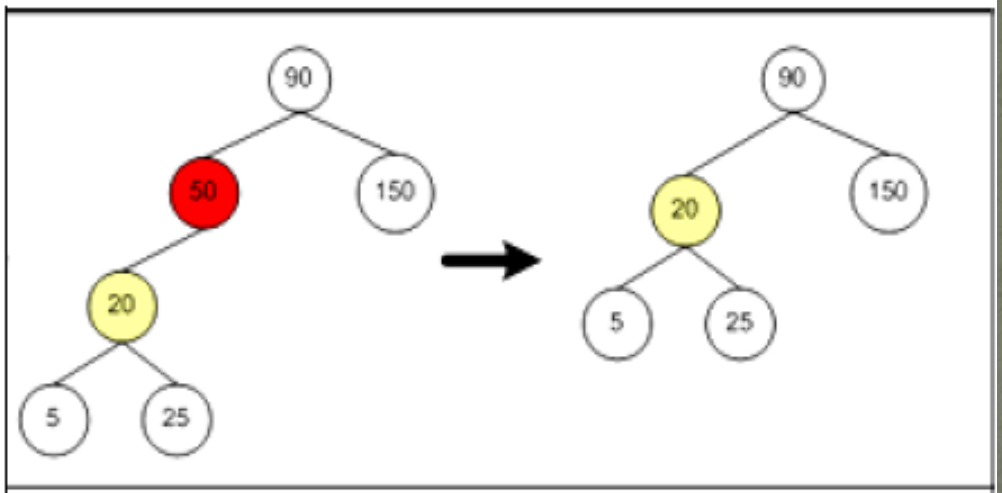
iii. valoarea căutată este mai **mare** decât cheia asociată nodului,
operaţia de ştergere continuă pentru subarborele **drept**

altfel
nu există în arbore o cheie egală cu valoarea dată,
iar operaţia de ştergere nu se poate efectua

Pentru ştergerea unui nod avem următoarele cazuri:

Cazul I

Dorim să ştergem nodul 50. Acesta nu are subarbore drept, aşa că îl vom înlocui pur şî simplu cu nodul 20.

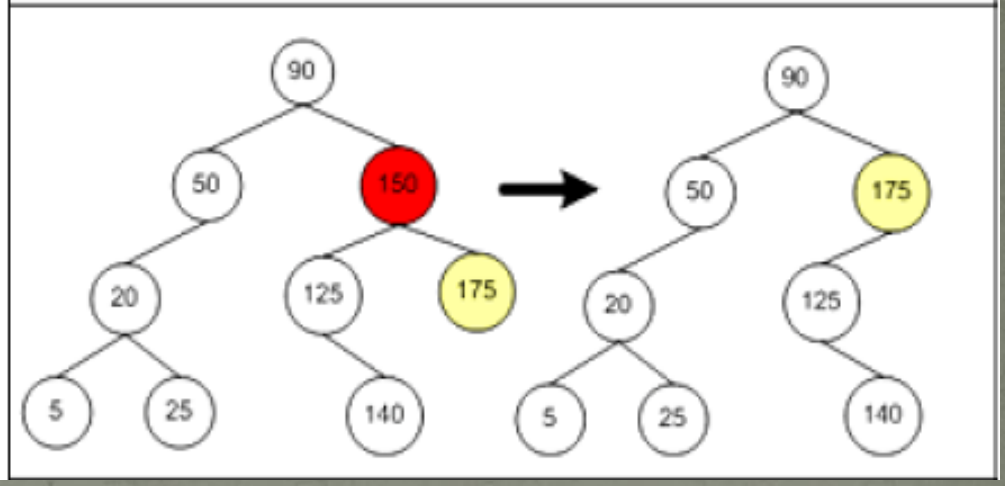


Cazul II

Dorim să ştergem nodul 150.

Subarborele drept nu conține decât nodul 175 Nu există un subarbore stâng al subarborelui drept.

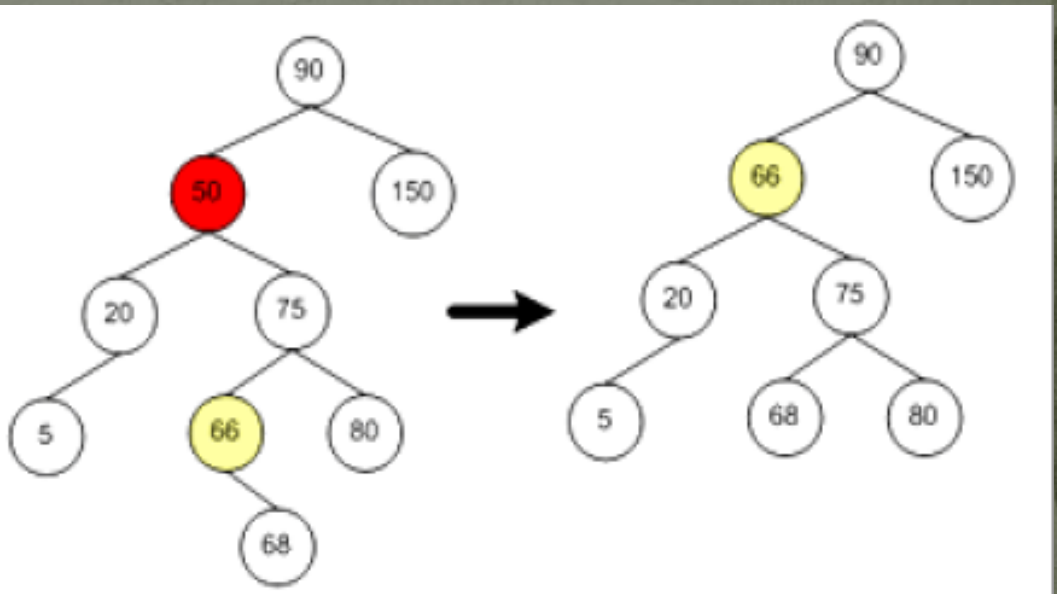
Se va înlocui nodul 150 cu 175.



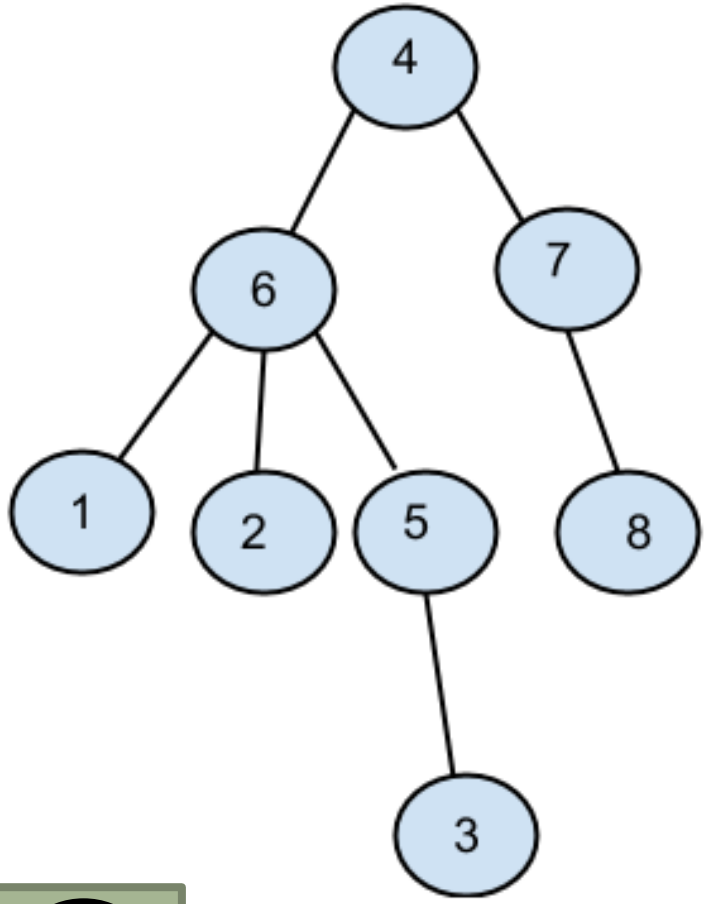
Cazul III

Dorim să ştergem nodul 50.

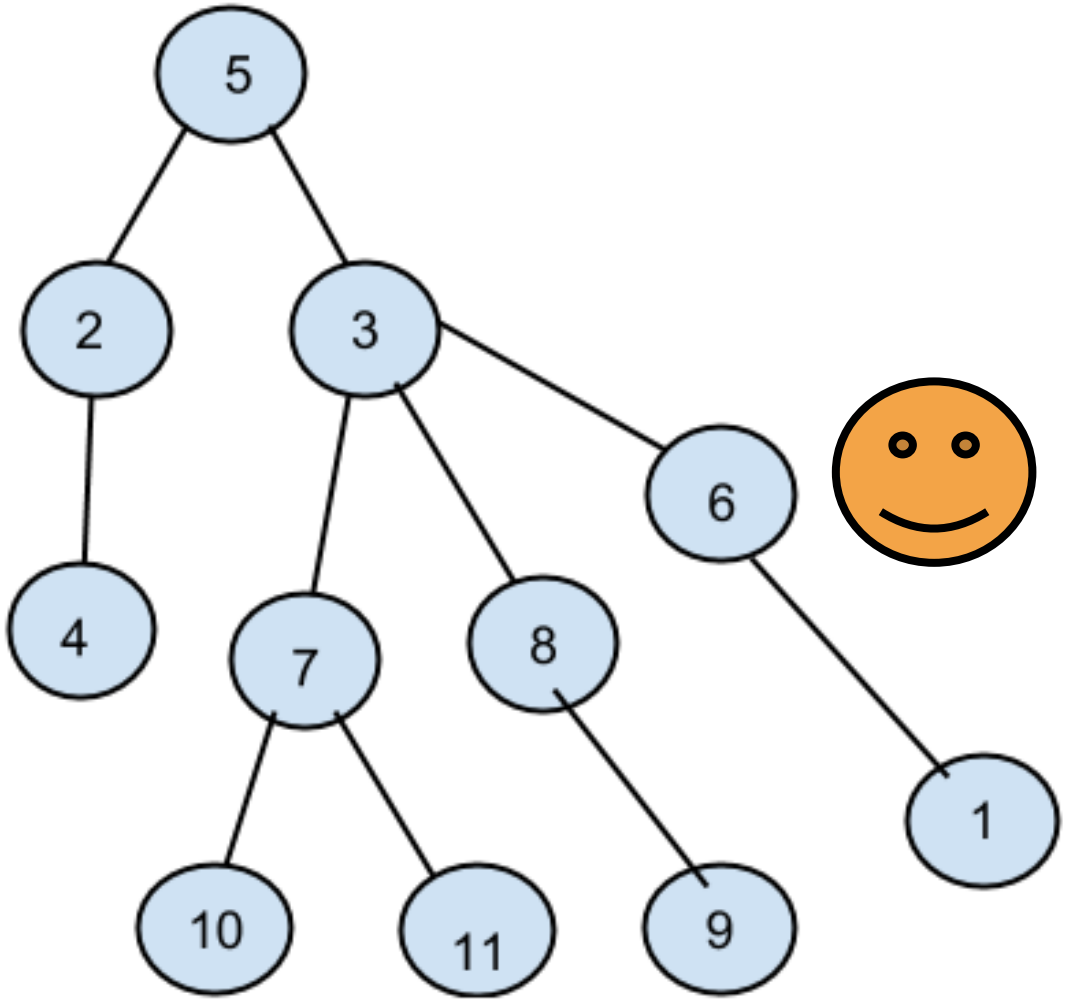
Deoarece subarborele drept al nodului 50 conține un subarbore stâng, se va alege cel mai din stânga nod al subarborelui drept a lui 50. Acest cel mai din stânga nod va conține *cel mai mic număr mai mare decât nodul de şters*. In cazul nostru acest nod este 66.



2.Prin înălţimea unui arbore cu rădăcină înţelegem numărul de muchii ale celui mai lung lanţ format din noduri distincte care are una dintre extremităţi în rădăcina arborelui. Scrieţi care este înălţimea şi care sunt frunzele arborelui descris prin următorul vector "de taţi":
1 2 3 4 5 6 7 8
(6,6,5,0,6,4,4,7).



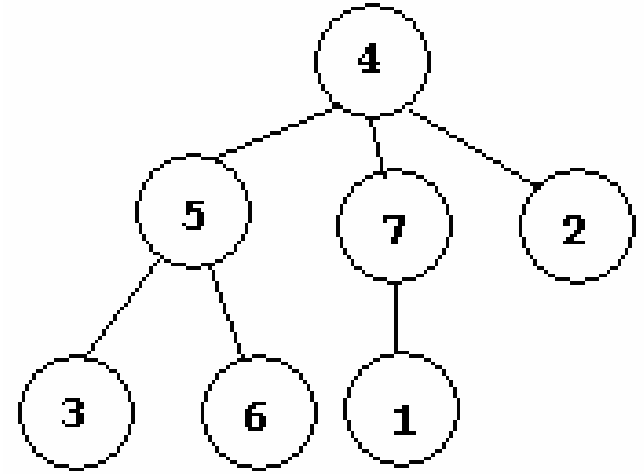
4. Câte frunze are arborele cu rădăcină descris prin următorul vector "de taţi":
1 2 3 4 5 6 7 8 9 10 11
(6,5,5,2,0,3,3,3,8, 7, 7)? (4p.)
a. 1 b. 2 c. 5 d. 4



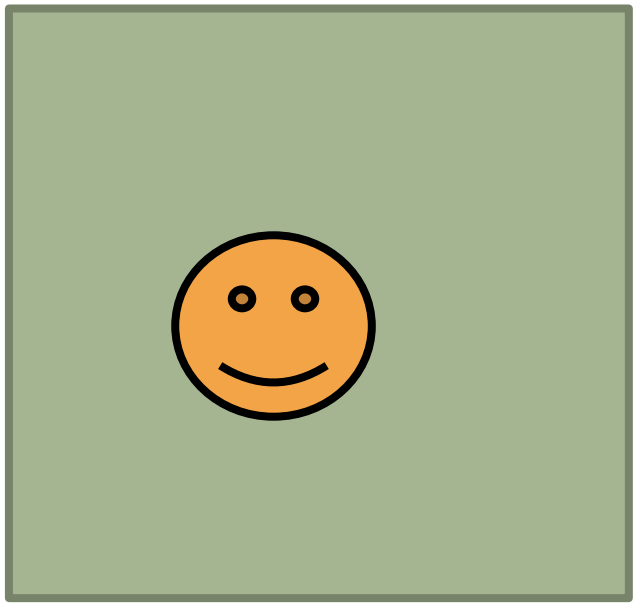
18. Care este vectorul "de taţi" pentru arborele cu rădăcină din figura alăturată?

a. 0 0 5 7 6 5 1 b. 1 0 0 7 6 5 0

c. 7 4 5 0 4 5 4 d. 7 4 5 0 4 5 7



3. Care sunt nodurile care au exact 2 descendenţi pentru un arbore cu rădăcină, cu 7 noduri, numerotate de la 1 la 7, dat de vectorul de "taţi": (3,3,0,1,2,2,4)?



Raspuns: 2,3

1.Se citeste un arbore cu n varfuri dat prin vectorul TATA.

a) Sa se afiseze muchiile arborelui

b) Sa se construiasca si sa se afiseze matricea de adiacenta a arborelui.

Ex: Pentru vectorul de tati 2 0 2 1 3 se vor afisa muchiile [1,2] [2,3] [1,4] [3,5] si matricea de adiacenta

0 1 0 1 0
1 0 1 0 0
0 1 0 0 1
1 0 0 0 0
0 0 1 0 0

```
#include<iostream.h>
int n, T[100], a[100][100];

void afis()
{ int i,j;
  for(i=1;i<=n;i++)
    { for(j=1;j<=n;j++)
          cout<<a[i][j]<<" ";
          cout<<endl;
    }
}

void main()
{ int i;
  cin>>n;
  for(i=1;i<=n;i++) cin>>T[i];
  for(i=1;i<=n;i++)
    if(T[i]!=0)
      { cout<<"["<<T[i]<<","<<i<<"] ";
        a[i][T[i]]=a[T[i]][i]=1;
      }
  cout<<endl;
  afis();
}
```

- Matricea de adiacenta asociatã unui graf neorientat este o matrice simetricã
- Suma elementelor de pe linia k reprezintã gradul nodului k
- Suma elementelor de pe coloana k reprezintã gradul nodului k
- Numarul de noduri este 6 si numarul de muchii este 5
- Matricea este simetrica si patratica avand 6 linii si 6 coloane
- Diagonala principala contine numai valori nule

- Pentru a prelucra graful se citesc:

6- reprezentand n, numarul de noduri

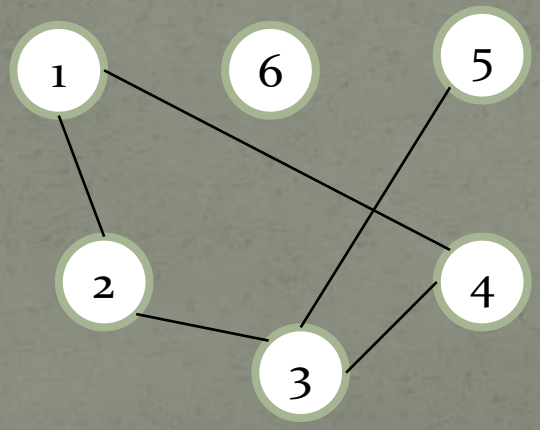
5- reprezentand m, numarul de muchii

5 perechi x-y reprezentand extremitatile celor 5 muchii:

- 1-2 => a[1,2]=a[2,1]=1
- 1-4 => a[1,4]=a[4,1]=1
- 2-3 => a[2,3]=a[3,2]=1
- 3-4=> a[3,4]=a[4,3]=1
- 3-5 => a[3,5]=a[5,3]=1

A=

0	1	0	1	0	0	nodul 1 are gradul 2
1	0	1	0	0	0	nodul 2 are gradul 2
0	1	0	1	1	0	nodul 3 are gradul 3
1	0	1	0	0	0	nodul 4 are gradul 2
0	0	1	0	0	0	nodul 5 are gradul 1
0	0	0	0	0	0	nodul 6 are gradul 0



2.Se citeste un arbore cu n varfuri dat prin vectorul muchiilor si apoi se citeste varful radacina. Sa se construiasca si sa se afiseze vectorul TATA.
Ex: Pentru un arbore cu 5 noduri, cu muchiile [1,2] [2,3] [1,4] [3,5] si radacina 2 se obtine vectorul de tati 2 0 2 1 3

Rezolvare:

```
#include<iostream.h>
int n, r, T[100], a[100][100], p[100];

void citire()
{ int i,x,y;
  cin>>n;
  for(i=1;i<=n-1;i++)
    { cin>>x>>y;
      a[x][y]=a[y][x]=1;;
    }
  cin>>r;
}

void BF(int r)
{  int s,d,i,x[100];
   d=s=1;
   x[1]=r; p[r]=1;
   while (s<=d)
   { for(i=1;i<=n;i++)
      if(a[x[s]][i] &&!p[i])
         { d++; x[d]=i;
           p[i]=1; T[i]=x[s];
         }
     s++;
   }
```

Heapuri:

- Un min-heap binar este un arbore binar în care fiecare nod are proprietatea că valoarea sa este mai mică sau egală decât cea a tuturor descendenților săi.
- h[parinte(x)] <= h[x]
- **h[x]** reprezintă valoarea nodului **x**, din vectorul h asociat arborelui.
- Un max-heap are semnul inegalității inversat
- Util in sortare rapida

Noțiuni de bază despre Red-Black Trees

- Un arbore rosu-negru este un arbore binar de cautare care are un bit suplimentar pentru memorarea fiecarui nod: culoarea acestuia, care poate fi rosu sau negru. Prin restrângerea modului în care se coloreaza nodurile pe orice drum de la radacina la o frunza, arborii rosu-negru garanteaza ca nici un astfel de drum nu este mai lung decât dublul lungimii oricarui alt drum, deci ca arborele este aproximativ echilibrat.

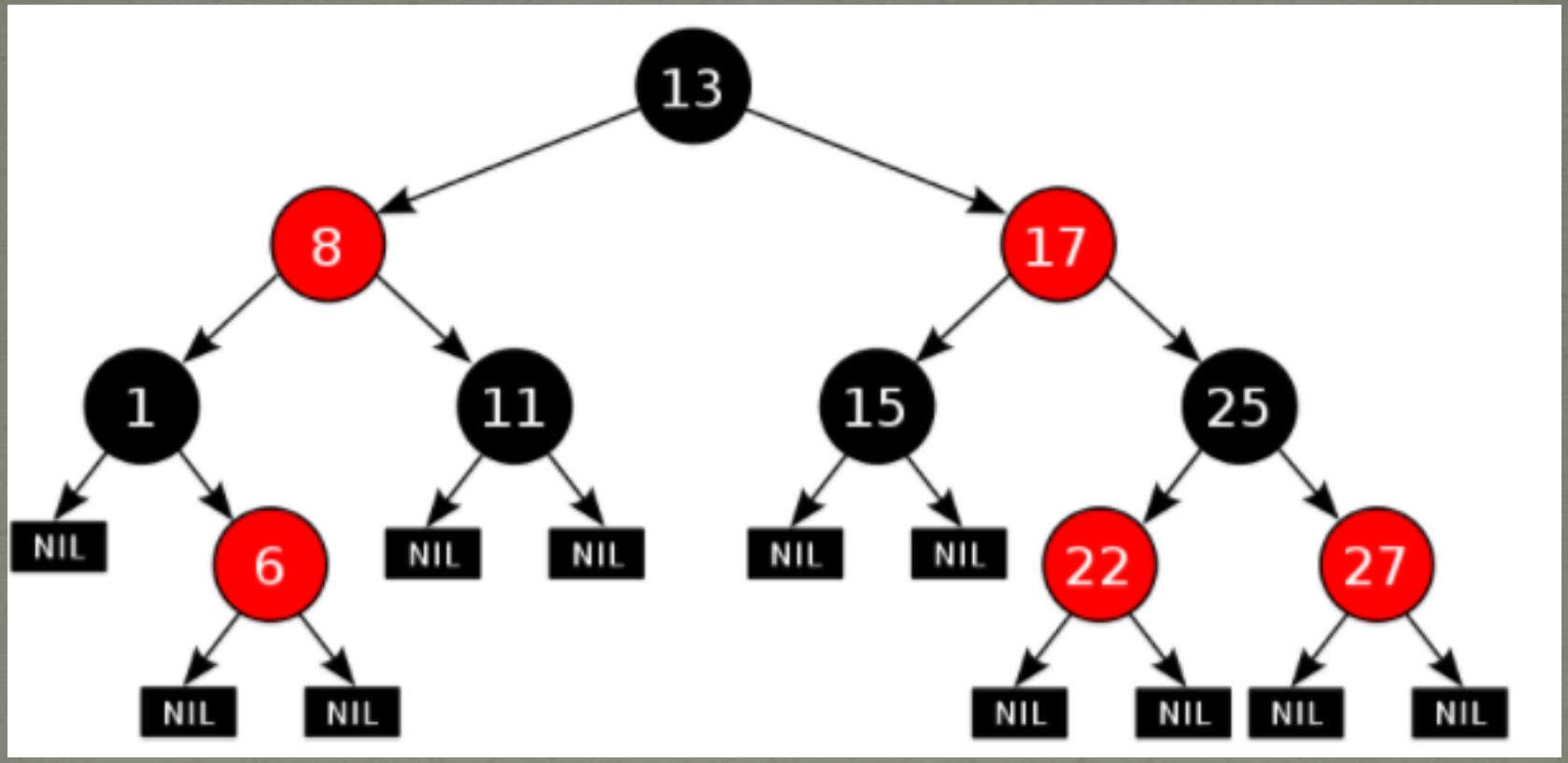
- Un arbore binar de cautare este arbore rosu-negru daca el îndeplineste urmatoarele proprietati rosu-negru:

  Fiecare nod este fie rosu, fie negru.

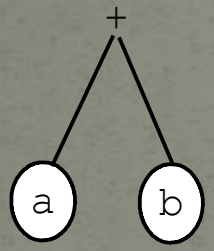
  Fiecare frunza (nil) este neagra.

  Daca un nod este rosu, atunci ambii fii ai sai sunt negri.

  Fiecare drum simplu de la un nod la un descendent care este frunza contine acelasi numar de noduri negre.


13
8
17
1
11
15
25
NIL
6
NIL
NIL
NIL
NIL
22
27
NIL
NIL
NIL
NIL
NIL
NIL
NIL
NIL

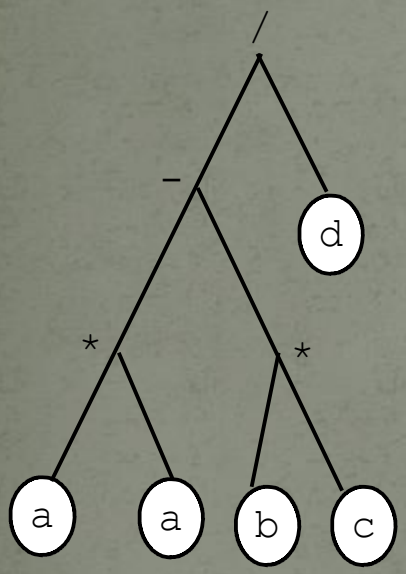
- Ex1: Sa se scrie programul de executie a operatiei x=a+b




```
READ a ; citeste valoarea lui a in acumulator
PUSH   ; salveaza in stiva
READ b ; citeste valoarea lui b in acumulator
PUSH   ; salveaza in stiva
ADD    ; aduna pe a cu b in stiva
POP    ; aduce suma in acumulator
WRITE x; salveaza in x
```

- Ex2: Sa se scrie programul de determinare a expresiei

E=(a^2-bc)/d




```
READ a ; citeste valoarea lui a in acumulator
PUSH   ; salveaza in stiva
READ a ; citeste valoarea lui a in acumulator
PUSH   ; salveaza in stiva
MULT   ; calculeaza a^2 in stiva
READ b ; citeste valoarea lui b in acumulator
PUSH   ; salveaza in stiva
READ c ; citeste valoarea lui c in acumulator
PUSH   ; salveaza in stiva
MULT   ; calculeaza b*c in stiva
SUB    ; scade din penultima valoare din stiva
         (a^2)pe ultima din stiva (b*c)
READ d ; citeste valoarea lui d in acumulator
PUSH   ; salveaza in stiva
DIV    ; imparte penultima valoare din stiva
         (a^2-b*c)la ultima (d)
POP    ; aduce rezultatul in acumulator
WRITE E; salveaza in E
```

- Ex3: Sa se scrie programul de determinare a expresiei

  E=(a+bc)/(a-bc)

- Dupa cum se poate remarca, procesoarele cu 0-adrese necesita multe instructiuni chiar si pentru operatii matematice simple. Ele sunt utilizate la SC cu procesoare de tip controler, care au un set mic de instructiuni elementare executabile.

- Acest tip de SCTR se gaseste incorporat intr-o multitudine de aparate casnice: masini de spalat automate, aspiratoare, etc.

Arhitecturi de 1-adresa

- Acest tip de procesoare pot utiliza ca zone de memorie pe langa registrul acumulator si o memorie adresabila sau un registru general.
- In campul operand al instructiunilor recunoscute de aceste arhitecturi poate exista maximum o adresa.
- Modurile de adresare pot fi cele implicite, immediate sau directe cu sau fara utilizarea registrilor.
- Operatiile aritmetice sau logice se efectueaza intre operandul aflat in acumulator si cel aflat intr-un registru sau la o adresa de memorie. Rezultatul operatiei ramane in acumulator.
- La aceste procesoare se pierde doar unul dintre operanzi, cel aflat initial in acumulator.
- Rezultatele partiale ale operatiilor se pastreaza in registrii generali.

- La sistemele de 1-adresa familia de instructiuni se largeste cu instructiuni aritmetice sau logice intre acumulator si un alt operand, instructiuni de salt si de test si instructiuni de apelare subrutine.

- Permit rezolvarea problemelor mai complexe decat determinarea unor expresii logice sau aritmetice.

- Exemplu:Sa se scrie programul de determinare a expresiei

  E=(a^2-bc)/d

```
READ d          ; citeste valoarea lui d in acumulator
STORE R3        ; salveaza in registru R3
READ b          ; citeste valoarea lui b in acumulator
STORE R1        ; salveaza in registru R1
READ c          ; citeste valoarea lui c in acumulator
MULT R1         ; calculeaza b*c in acumulator
STORE R1        ; salveaza b*c in R1
READ a          ; citeste valoarea lui a in acumulator
STORE R2        ; salveaza in registrul R2
MULT R2         ; calculeaza pe a^2 in acumulator
SUB R1          ; scade din a^2 pe bc in acumulator
DIV R3          ; imparte pe (a^2-b*c) din acumulator la
                  d din R3
WRITE E         ; salveaza in E
```

- Numarul de instructiuni a scazut fata de acelasi exemplu rezolvat cu sistemul de 0-adresa, ceea ce nu inseamna ca executia este mai rapida, avand in vedere ca o mare parte din instructiunile folosite sunt cu adresare directa deci utilizeaza mai multi cicli de masina pentru incarcarea instructiunii.

Arhitecturi de 2-adrese

- Astfel de SC utilizeaza instructiuni care permit descrierea in campul operand a doua adrese distincte, care pot fi adrese de registri sau adrese de memorie iar registrul de acumulator nu este considerat ca o adresa implicita.
- Programele scrise pentru sistemele de 2-adrese utilizeaza acelasi categorii de instructiuni ca cele de 1-adresa, cu diferenta ca operatiile algebrice sau logice nu se efectueaza intre acumulator si o locatie de memorie ci intre doua locatii de memorie. Nu este nevoie sa se salveze in permanenta informatiile din acumulator.

- Este legal în Romania, ca un bărbat să se însoare cu sora văduvei sale?
- R: