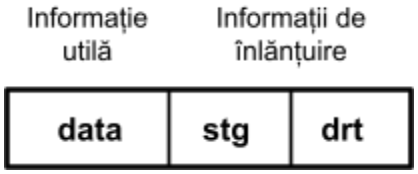
Arbori binari

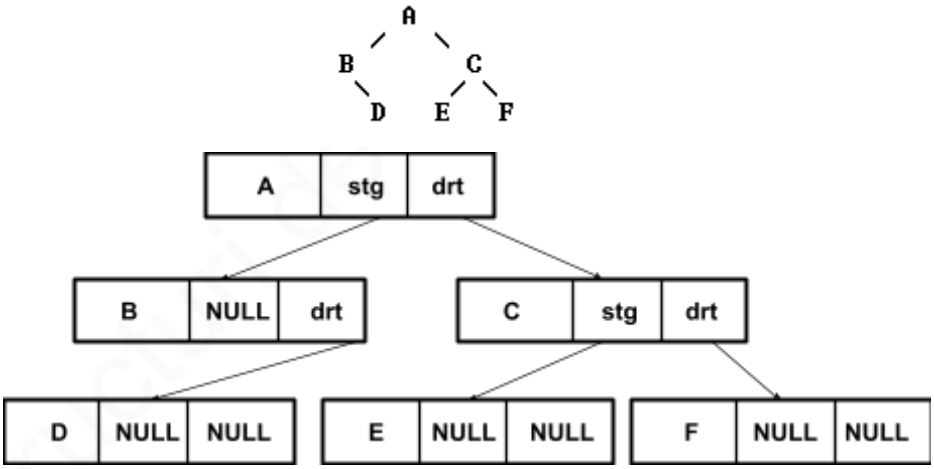
1. Reprezentarea standard

În reprezentarea standard, un nod al arborelui este o structură cu un câmp conținând eticheta nodului (data) și două câmpuri pointeri la cei doi descendenți (lchild și rchild):




```C/C++
struct Nod{
   type data;
   Nod* stg, *drt;
};
```

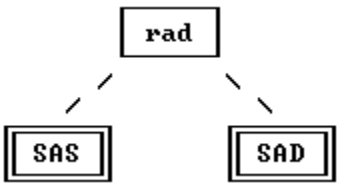
Astfel, arborele:



Pentru a putea prelucra un arbore este suficient să cunoaștem un pointer la nodul rădăcină. Valoarea **NULL** pentru acest pointer va semnifica un arbore vid.

2. Parcurgeri

Un arbore binar poate fi privit conform următoarei scheme recursive:

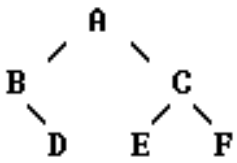


rad = rădăcina
SAS = SubArbore Stâng
SAD = SubArbore Drept

Pe aceasta schemă se definesc cele trei moduri de parcurgere a arborelui:

1. **PREORDINE** : rad SAS SAD
   Se prelucrează mai întâi rădăcina apoi se parcurg în preordine subarborii stâng și drept.
2. **INORDINE** : SAS rad SAD
   Se parcurge în inordine subarborele stâng, se prelucrează rădăcina și apoi se parcurge în inordine subarborele drept.
3. **POSTORDINE** : SAS SAD rad
   Se parcurg mai întâi în postordine subarborii stâng și drept apoi se prelucrează rădăcina.

Pentru arborele:



cele trei parcurgeri prelucrează nodurile în ordinea:

PREORDINE: A B D C E F
INORDINE: B D A E C F
POSTORDINE: D B E F C A

Putem realiza aceste parcurgeri utilizând subrutine recursive. De exemplu:

```
C/C++
void PREORDINE(Nod* p){
	if (p!=NULL){
		prelucreaza(*p);
		PREORDINE(p->stg);
		PREORDINE(p->drt);
	}
}

//SAU

void PREORDINE(Nod* p){
	prelucreaza(*p);
	if(p->stg!=NULL) PREORDINE(p->stg);
	if(p->drt!=NULL) PREORDINE(p->drt);
}
```

A doua varianta nu poate fi aplicată unui arbore vid, în timp ce prima tratează corect arborele vid, în schimb execută un apel recursiv în plus pentru fiecare legătură care este NULL.

3. Exemplu - Calcularea valorii maxime dintr-un arbore

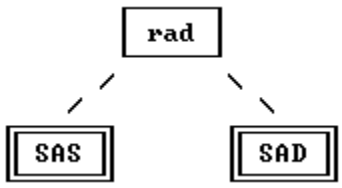
Varianta 1

```
C/C++
char max ;     // max este variabila globala

void CautaMax(Nod* p)
{/* ----------------Parcurgere preordine Varianta 2*/
      if (p!=NULL){
          if (p->data>max) max=p->data;
          CautaMax(p->stg);
          CautaMax(p->drt);
      }
}

char ValMax(Nod* p)
{
      max = 0;
      CautaMax(rad);
      return max;
}
```

Funcția **ValMax** apelează o procedură recursivă **CautaMax** care face o parcurgere prin arbore testând valoarea fiecărui nod. La sfârșitul parcurgerii, variabila "*max*", care este o variabila globală (externă) pentru procedura recursivă, și care a fost inițializată cu cea mai mică valoare de tip *char*, va conține valoarea maximă a etichetelor din arbore.

Varianta 2

Pornind de la schema:



stabilim următoarea definiție recursivă:

ValMax(arbore) = max(rad, ValMax(SAS), ValMax(SAD))

```
C/C++
//Varianta 2
char max(char v1,char v2)
{
     if(v1>=v2) return v1;
     else return v2;
}

char ValMax(Nod* rad)
{
```

```
        char vmax;

            vmax = rad->data;
            if rad->stg!=NULL
                        vmax = max(vmax,ValMax(rad->stg));
            if rad->drt!=NULL
                        vmax = max(vmax,ValMax(rad->drt));
            return vmax;
}
```

Această variantă nu se poate aplica unui arbore vid, dar are avantajul ca se poate aplica și în cazuri în care nu există o valoare de etichetă pentru nod mai mică decât toate etichetele posibile (cum am folosit mai sus, valoarea 0).

3. Aplicații

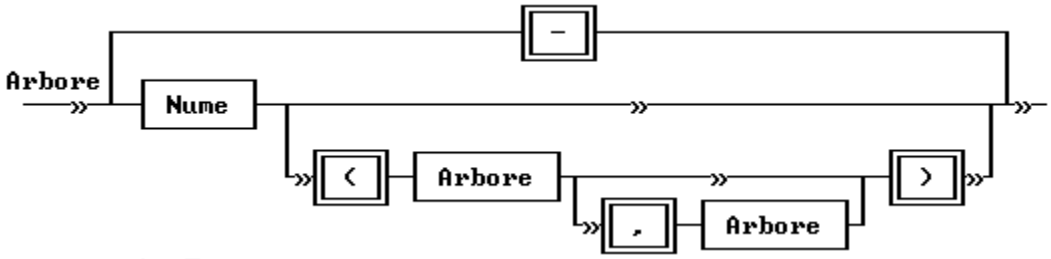
Modulul ARBORE.CPP (vezi Anexa) conține declarațiile tipurilor:

```
struct Nod {
  char data;
  struct Nod *stg, *drt;
 }
```

și functia:

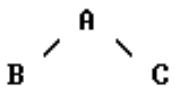
```
Nod* creareArbore();
```

care citește un arbore specificat conform următoarei diagrame de sintaxă, și întoarce pointer la rădăcina arborelui citit.

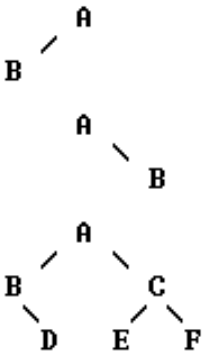


În diagramă: '-' -semnifică un arbore vid;
nume -este eticheta unui nod formată dintr-o literă.

Exemple: Arborele vid: -



A(B,C)



A(B) sau A(B,-)

A(-,B)

A(B(-,D),C(E,F))

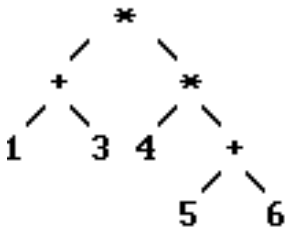
1. Să se scrie și să se testeze următoarele subrutine. Încercați pe rând, pentru fiecare, cele două variante de abordare prezentate în exemplul cu aflarea valorii maxime):

a) Să se afișeze conținutul arborelui în INORDINE.
b) Să se afișeze conținutul arborelui în POSTORDINE.
c) O funcție pentru determinarea adâncimii arborelui.
d) O funcție pentru determinarea numărului de noduri din arbore.
e) O funcție pentru determinarea numărului de frunze ale arborelui.
f) Să se afișeze toate nodurile care au valoarea din rădăcină mai mare decât toate valorile din subarborii descendenți.
g) Să se afișeze toate nodurile pentru care toate valorile conținute în subarborele stâng sunt mai mici decât toate valorile conținute în subarborele drept.
h) Pentru fiecare nod să se comute subarborele stâng cu cel drept și să se afișeze conținutul arborelui în forma cu paranteze.

2. Să se scrie un program care citește expresii formate din operanzi, numere întregi de o cifră și operatorii + și *.

a) Să se creeze arborele expresiei;
Indicație: Se crează arborele din forma postfixată a expresiei, folosind o stivă de (pointeri la) noduri. Se creează (sub)arbori care se combină treptat într-un singur arbore final.
b) Să se calculeze valoarea expresiei pe arbore;
c) Să se afișeze expresia în forma prefixată.

De exemplu arborele corespunzător expresiei: **(1+3)*4*(5+6)**
este:



Structuri de Date și Algoritmi – Laborator 9

Notare:
Problema 1 - a) 1p ; b) 1p ; c) 1p ; d) 1p ; e) 1p ; f) 1.5p ; g) 1.5p ; h) 2p
Problema 2 - a) 1p ; b) 2p ; c) 1p

Aplicațiile neterminate în timpul orelor de laborator rămân ca teme pentru studiu individual!

Structuri de Date și Algoritmi