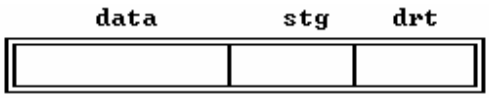
Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 9

Arbori binari

1. **Reprezentarea standard**
2. **Parcurgeri**
3. **Exemplu: calcularea valorii maxime dintr-un arbore**

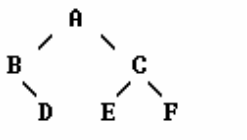
1. Reprezentarea standard

In reprezentarea standard, un nod al arborelui este o structura cu un cimp continind eticheta nodului (*data*) si doua cimpuri pointeri la cei doi descendenti (*lchild* si *rchild*):

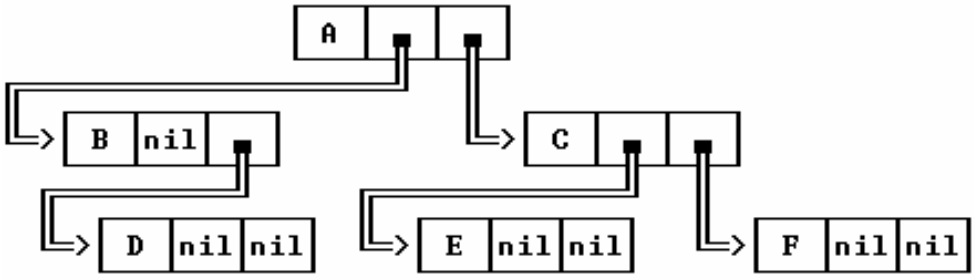



```
struct Nod
  {
        type data;
        Nod* stg, *drt;
  };
```

Astfel, arborele:



va avea urmatoarea reprezentare:

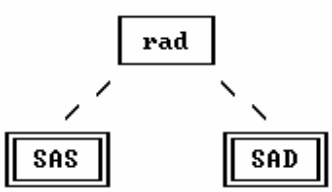


Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 9

Pentru a putea prelucra un arbore este suficient sa cunostem un pointer la nodul radacina. Valoarea *nil* pentru acest pointer va semnifica un arbore vid.

2. Pacurgeri

Un arbore binar poate fi privit conform urmatoarei scheme recursive:



rad = radacina
SAS = SubArbore Sting
SAD = SubArbore Drept

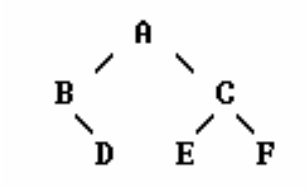
Pe aceasta schema se definesc cele trei moduri de parcurgere a arborelui:

PREORDINE : rad SAS SAD
Se prelucreaza mai intii radacina apoi se parcurg in preordine subarborii sting si drept.

INORDINE : SAS rad SAD
Se parcurge in inordone subarborele sting, se prelucreaza radacina si apoi se parcurge in inordine subarborele drept.

POSTORDINE : SAS SAD rad
Se parcurg mai intii in postordine subarborii sting si drept apoi se prelucreaza radacina.

Pentru arborele:



cele trei parcurgeri prelucreaza nodurile in 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 utilizind subrutine recursive. De exemplu:

```
void PREORDINE(pNod p);
{
     if (p!=NULL){
        prelucreaza(*p);
       PREORDINE(p->lchild);
       PREORDINE(p->rchild);
       }
}
```

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 aplicata unui arbore vid, in timp ce prima trateaza corect arborele vid, in schimb executa un apel recursiv in plus pentru fiecare legatura care este NULL.

3. Exemplu - Calcularea valoarii maxime dintr-un arbore.

Varianta 1

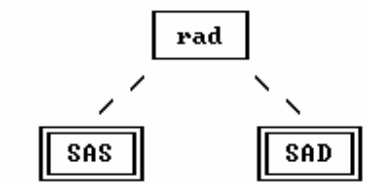
```
char max ;     // max este variabila globala

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;
}
```

Functia `Valmax` apeleaza o procedura recursiva `CautaMax` care face o parcurgere prin arbore testind valoarea fiecarui nod. La sfirsitul parcurgerii, variabila "*max*", care este o variabila globala (externa) pentru procedura recursiva, si care a fost initializata cu cea mai mica valoare de tip *char*, va contine valoarea maxima a etichetelor din arbore.

Varianta 2

Pronind de la schema:



stabilim urmatoarea definitie recursiva:

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

Apelurile `ValMax(SAS)` si `ValMax(SAD)` se vor executa numai daca subarborii nu sint vizi.

Iata implementarea:

```
char max(char v1,char v2)
 {
     if(v1>=v2) return v1;
     else return v2;
 }

char ValMax(pNod rad)
{
 char vmax;

     vmax = rad->data;
     if rad->lchild!=NULL
                vmax = max(vmax,ValMax(rad->lchild));
     if rad->rchild!=NULL
                vmax = max(vmax,ValMax(rad->rchild));
```

```
        return vmax;
    }
```

Aceasta varianta nu se poate aplica unui arbore vid, dar are avantajul ca se poate aplica si in cazuri in care nu exista o valoare de eticheta pentru nod mai mica decit toate etichetele posibile (cum am folosit mai sus, valoarea 0).

TEMA

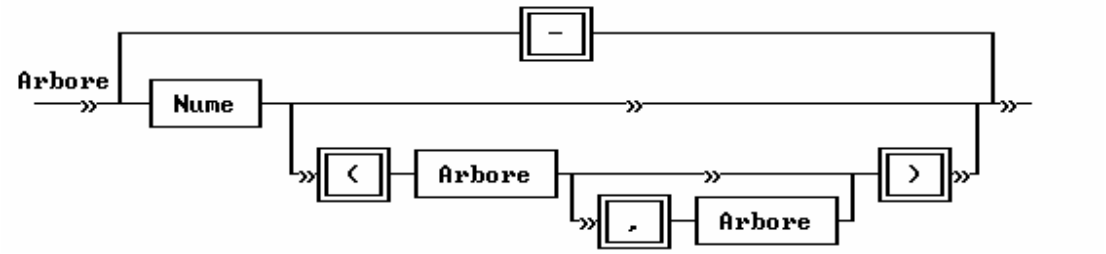
1. Modulul ARBORE.CPP (vezi Anexa) contine declaratiile tipurilor:

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

si functia:

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

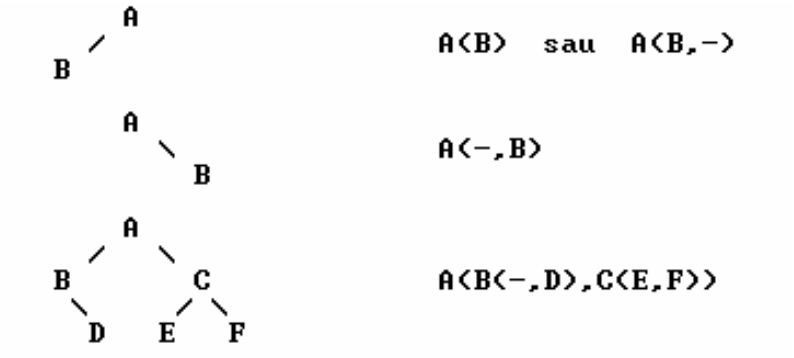
care citeste un arbore specificat conform urmatoarei diagrame de sintaxa, si intoarce pointer la radacina arborelui citit.



In diagrama: '-' -semnifica un arbore vid;
nume -este eticheta unui nod formata dintr-o litera.

Exemple: Arborele vid: -

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 9

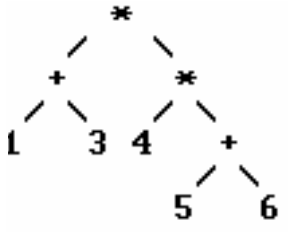


Sa se scrie si sa se testeze urmatoarele subrutine. Incercati pe rind, pentru fiecare, cele doua variante de abordare prezentate in exemplul cu aflarea valorii maxime):

- Sa se afiseze continutul arborelui in INORDINE.
- Sa se afiseze continutul arborelui in POSTORDINE.
- O functie pentru determinarea adincimii arborelui.
- O functie pentru determinarea numarului de noduri din arbore.
- O functie pentru determinarea numarului de frunze ale arborelui.
- Sa se afiseze toate nodurile care au valoarea din radacina mai mare decit toate valorile din subarborii descendenti.
- Sa se afiseze toate nodurile pentru care toate valorile continute in subarborele sting sint mai mici decit toate valorile continute in subarborele drept.
- Pentru fiecare nod sa se comute subarborele sting cu cel drept si sa se afiseze continutul arborelui in forma cu paranteze.

2. Sa se scrie un program care citeste expresii formate din operanzi, numere intregi de o cifra, si operatorii + si *. Creeaza arborele expresiei si calculeaza valoarea expresiei pe arbore.

De exemplu expresia: **(1+3)*4*(5+6)**
este citita in arborele:



Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 9

ANEXA

Arbore.h

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

Nod* creareArbore();
```

Arbore.cpp

```
#include <alloc.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include "arbore.h"

void eroare();
char readchar();
char citesteNume();
Nod* citesteArbore();
Nod* creareArbore();

char car;

void eroare()
{
     printf("Sirul de intrare este eronat!\n");
     printf("Apasati tasta o tasta...");
     getch();
     exit(1);
}

char readchar()
{
     char c;
     do  c=getchar();  while(c==' ');
     return c;
}

char citesteNume()
{
     char c;
```

```
     if(!isalpha(car)) eroare();
     c = car;
     car = readchar();
     return c;
}

Nod* citesteArbore()
{
     Nod* rad;
     if( car=='-' )  {
               rad=0;
               car = readchar();
               }
     else {
       rad = (Nod*) malloc(sizeof(Nod));
       rad->data = citesteNume();
       if( car!='(' )  {
                rad->stg = 0;
                rad->drt = 0;
                }
       else {
            car = readchar();
            rad->stg = citesteArbore();
            if( car!=',' )    rad->drt = 0;
            else {
              car = readchar();
              rad->drt = citesteArbore();
              }
            if( car!=')' )  eroare();
            car = readchar();
            }
       }
     return rad;
}

Nod* creareArbore()
{
     printf("\nIntroduceti arborele:");
     car = readchar();
     return citesteArbore();
}
```