ARBORI

I. ASPECTE TEORETICE

Arborii, ca si listele, sunt structuri de date de natura recursiva si dinamica. Prin **arbore** intelegem o multime finita si nevida de elemente numite noduri:

ARB = { A1, A2, A3, ..., An }, unde n > 0

care au urmatoarele proprietati:

- Exista un nod si numai unul care se numeste **radacina** arborelui.
- Celelalte noduri formeaza submultimi disjuncte ale lui A, care formeaza fiecare cate un arbore. Arborii respectivi se numesc **subarbori** ai radacinii.

Intr-un arbore exista noduri carora nu le mai corespund subarbori. Un astfel de nod se numeste **terminal** sau **frunza**. In legatura cu arborii s-a stabilit un limbaj conform caruia un nod radacina se spune ca este un nod **tata**, iar subarborii radacinii sunt **descendentii** acestuia. Radacinile descendentilor unui nod tata sunt **fiii** lui.

Arborii binari

Un **arbore binar** este o multime finita de elemente care sau este **vida**, sau contine un element numit **radacina**, iar celelalte elemente se impart in doua submultimi disjuncte, care fiecare la randul ei, este un arbore binar. Una din submultimi este numita **subarborele stang** al radacinii, iar cealalta **subarborele drept**. Arborele binar este ordonat, deoarece in fiecare nod, subarborele stang se considera ca precede subarborele drept. De aici rezulta ca un nod al unui arbore binar are cel mult doi fii si ca unul este **fiul stang**, iar celalalt este **fiul drept**. Fiul stang este mai in varsta decat cel drept. Un nod al unui arbore binar poate sa aiba numai un singur descendent. Acesta poate fi subarborele stang sau subarborele drept.

Obs.: Cele doua posibilitati se considera distincte. Cu alte cuvinte, daca doi arbori binari difera numai prin aceea ca nodul A dintr-un arbore are ca descendent numai fiul stang, iar acelasi nod din celalalt arbore are ca descendent numai fiul drept, cei doi arbori se considera distincti.

Un arbore binar nu se defineste ca un caz particular de arbore ordonat. Astfel, un arbore nu este niciodata vid, spre deosebire de un arbore binar care poate fi si vid.

Orice arbore ordonat poate fi intotdeauna reprezentat printr-un arbore binar. Avand in vedere acest aspect, in continuare ne vom ocupa numai de arbori binari.

Nodul unui arbore binar poate fi reprezentat ca o data structurala de tipul NOD care se defineste in felul urmator:

```
typedef struct nod
                    {
                     declaratii
                     struct nod *st;
                     struct nod *dr;
                    }  NOD;
```

unde:

- st - este pointerul spre fiul stang al nodului curent;
- dr - este pointerul spre fiul drept al aceluiasi nod.

Asupra arborilor binari se pot defini mai multe operatii dintre care amintim:

- inserarea unui nod frunza intr-un arbore binar;
- accesul la un nod al unui arbore;
- parcurgerea unui arbore;

- stergerea unui arbore.

Operatiile de **inserare** si **accesul** la un nod, au la baza un criteriu care sa defineasca locul in arbore al nodului in cauza. Acest criteriu este dependent de problema concreta la care se aplica arborii binari pentru a fi rezolvata. El va fi definit in partea de DESFASURAREA LUCRARII, printr-o functie pe care o vom denumi functia **criteriu**. Ea are doi parametri care sunt pointeri spre tipul NOD. Fie p1 primul parametru al functiei **criteriu** si p2 cel de-al doilea parametru al ei. Atunci, functia **criteriu** returneaza:

-1 - daca p2 pointeaza spre o data de tip NOD care poate fi un nod al subarborelui stang al nodului spre care pointeaza p1;

1 - daca p2 pointeaza spre o data de tip NOD care poate fi un nod al subarborelui drept al nodului spre care pointeaza p1;

0 - daca p2 pointeaza spre o data de tip NOD care nu poate fi nod al subarborilor nodului spre care pointeaza p1.

La construirea unui arbore se stabileste un criteriu pentru determinarea pozitiei in care sa se insereze in arbore nodul curent (nodul corespunzator valorii curent citite).

Spre exemplu, se considera:

a. **p1** - este pointer spre un nod al arborelui in care se face inserarea (initial p1 pointeaza spre radacina arborelui).

b. **p2** - este un pointer spre nodul curent (nodul de inserat).

c. daca **p2->val < p1->val**, atunci se va incerca inserarea nodului curent in subarborele stang al nodului spre care pointeaza p1.

d. daca **p2->val > p1->val**, atunci se va incerca inserarea nodului curent in subarborele drept al nodului spre care pointeaza p1.

e. daca **p2->val = p1->val**, atunci nodul curent nu se mai insereaza in arbore deoarece exista deja un nod corespunzator valorii curent citite

Atributiile ce semnaleaza un astfel de criteriu pot fi indeplinite (cum se va vedea si in DESFASURAREA LUCRARII) de catre o functie specializata (functia **criteriu**).

Nodul curent nu se mai insereaza in arbore in cazul descris la punctul **e** (situatie care in general corespunde cazului cand functia criteriu returneaza valoarea zero). In acest caz, nodurile spre care pointeaza p1 si p2 le vom considera **echivalente**.

De obicei, la intalnirea unei perechi de noduri echivalente, nodul din arbore (spre care pointeaza p1) este supus unei prelucrari, iar nodul curent (spre care pointeaza p2) este eliminat. Pentru a realiza o astfel de prelucrare este necesar sa se apeleze o functie care are ca parametri pointerii p1 si p2 si care returneaza un pointer spre tipul NOD (de obicei se returneaza valoarea lui p1). Vom numi aceasta functie **echivalenta**. Ea este dependenta de problema concreta, ca si functia **criteriu**.

Functii de tipul **incnod** si **elibnod** apelate pentru a incarca datele in nodul curent care urmeaza a fi inserat in arbore (sau prelucrat), respectiv pentru a elibera zona de memorie ocupata de un nod, sunt functii foarte asemanatoare cele intalnite in cadrul laboratorului referitor la liste.

In afara de functiile enumerate mai sus, vom folosi niste functii specifice pentru operatii asupra arborilor. Acestea utilizeaza utilizeaza o variabila globala care este un pointer spre radacina arborelui. Numim **prad** aceasta variabila. Ea se defineste astfel:

NOD *prad;

In cazul in care intr-un program se prelucreaza simultan mai multi arbori, nu se va utiliza variabila globala **prad**, interfata dintre functii realizandu-se cu ajutorul unui parametru care este pointer spre tipul NOD si caruia i se atribuie, la apel, adresa nodului radacina al arborelui prelucrat prin functia apelata.

Obs.: In continuare vom folosi functii care utilizeaza variabila globala **prad**.

Functia **insnod** insereaza un nod in arbore, conform urmatorilor pasi:

1. Se aloca zona de memorie pentru nodul care urmeaza sa se insereze in arbore. Fie **p** pointerul care are ca valoare adresa de inceput a zonei respective.

2. Se apeleaza functia **incnod**, cu parametrul p, pentru a incarca datele curente in zona spre care pointeaza p. Daca **incnod** retur-neaza neaza valoarea 1, se trece la pasul 3. Altfel se revine din functie cu valoarea zero.

3. Se fac atribuirile:

p->st = p->dr = 0

deoarece nodul de inserat este nod frunza.

4. **q = prad**

5. Se determina pozitia, in arbore, in care sa se faca inserarea. In acest scop se cauta nodul care poate fi nod tata pentru nodul curent:

i = criteriu(q,p)

6. Daca **i<0**, se trece la pasul 7; altfel se trece la pasul 8.

7. Se incearca inserarea nodului spre care pointeaza **p** (nodul curent) in subarborele stang al arborelui spre care pointeaza **q**.
* Daca **q -> st** are valoarea zero, atunci nodul spre care pointeaza q nu are subarbore stang si nodul curent devine fiu stang al celui spre care pointeaza q (q->st = p). Se revine din functie returnandu-se valoarea lui p.
Altfel se face atribuirea **q = q->st** (se trece la fiul stang al nodului spre care pointeaza q) si se trece la pasul 5.

8. Daca **i>0**, se trece la pasul 9; altfel se trece la pasul 10.

9. Se incearca inserarea nodului spre care pointeaza **p** (nodul curent) in subarborele drept al arborelui spre care pointeaza **q**.
* Daca **q -> dr** are valoarea zero, atunci nodul spre care pointeaza q nu are subarbore drept si nodul curent devine fiu drept al celui spre care pointeaza q (q->dr = p). Se revine din functie returnandu-se valoarea lui p.
Altfel se face atribuirea **q = q->dr** (se trece la fiul drept al nodului spre care pointeaza q) si se trece la pasul 5.

10. Nodul curent nu poate fi inserat in arbore. Se apeleaza functia **echivalenta** si se revine din functie cu valoarea returnata de functia **echivalenta**.

Prelucrarea informatiei pastrata in nodurile unui arbore binar se realizeaza parcurgand nodurile arborelui respectiv. Parcurgerea nodurilor unui arbore binar se poate face in mai multe moduri, dintre care se remarca:
- parcurgerea in preordine;
- parcurgerea in inordine;
- parcurgerea in postordine.

Parcurgerea in preordine inseamna accesul la radacina si apoi parcurgerea celor doi subarbori ai ei, intai subarborele stang, apoi cel drept. Subarborii, fiind ei insisi arbori binari, se parcurg in acelasi mod.

Parcurgerea in inordine inseamna parcurgerea mai intai a subarborelui stang, apoi accesul la radacina si in continuare parcurgerea subarborelui drept. Cei doi subarborii se parcurg in acelasi mod.

Parcurgerea in postordine inseamna parcurgerea mai intai a subarborelui stang, apoi a subarborelui drept si in final accesul la radacina arborelui. Cei doi subarborii se parcurg in acelasi mod.

Accesul la un nod permite prelucrarea informatiei continute in nodul respectiv. In acest scop se poate apela o functie care este dependenta de problema concreta care se rezolva cu ajutorul parcurgerii arborelui (de exemplu functia **prelucrare** din partea a doua a lucrarii).

II. DESFASURAREA LUCRARII

Se va alcatui un program care citeste cuvintele dintr-un text si afiseaza numarul de aparitii al fiecarui cuvant din textul respectiv. Cuvantul se defineste ca o succesiune de litere mici si/sau mari. Textul se termina prin sfarsit de fisier (CTRL+Z).

Aceasta problema a mai fost rezolvata intr-o sedinta anterioara, folosind o lista simplu inlantuita. Programul de fata rezolva aceasta problema folosind arbori binari.

NOTA: Functiile sau programele de la fiecare punct din desfasurarea lucrarii se vor scrie in fisiere separate cu extensia **.cpp** .

1. Sa se scrie o functie care citeste un cuvant si-l pastreaza in memoria heap (prin cuvant intelegandu-se o succesiune de litere mici sau mari). Functia returneaza adresa de inceput a zonei din memoria heap in care se pastreaza cuvantul citit sau zero la intalnirea sfarsitului de fisier (EOF sau Ctrl Z).

O posibila solutie:

```
char *citcuv()
/* - citeste un cuvant si-l pastreaza in memoria heap;
   - returneaza pointerul spre cuvantul respectiv sau zero la sfarsit de fisier. */
{
 int c, i;
 char t[255];
 char *p;

 /*salt peste caracterele care nu sunt litere */
 while((c=getchar()) < 'A' || (c > 'Z' &&
              c < 'a') || c > 'z')
            if(c == EOF)
                     return 0; /* s-a tastat EOF */

 /* se citeste cuvantul si se pastreaza in t */
 i=0;
 do
   {
    t[i++] = c;
   } while(((c=getchar()) >= 'A' && c <= 'Z' ||
                 c >= 'a') && c <= 'z');
 if(c == EOF)
    return 0;
 t[i++] = '\0';

 /* se pastreaza cuvantul in memoria heap */
 if((p = (char *)malloc(i)) == 0)
            {
             printf("memorie insuficienta\n");
             exit(1);
            }
 strcpy(p,t);
 return p;
}
```

Observatie: Modul de lucru al functiei **strcpy** va fi inteles prin studierea help-ului din C (CTRL + F1).

2. Se considera tipul utilizator:

```
typedef struct nod
                                    {
                                     char *cuvant;
                                     int frecventa;
                                     struct nod *st;
                                     struct nod *dr;
                                    }NOD
```

care va fi utilizat in toate punctele urmatoare de la desfasurarea lucrarii.

Se cere sa se scrie o functie **incnod** care incarca datele curente intr-un nod de tip NOD. Functia de fata apeleaza functia **citcuv** si atribuie adresa returnata de ea pointerului cuvant din nodul curent. De asemenea, se atribuie valoarea 1 variabilei frecventa. Functia incnod returneaza valoarea -1 daca **citcuv** returneaza valoarea zero si 1 altfel.

O posibila solutie:

```
int incnod(NOD *p)
/* incarca datele curente in nodul spre care pointeaza p */
{
 if((p -> cuvant = citcuv()) == 0)  return -1;
 p -> frecventa = 1;
 return 1;
}
```

3) Sa se scrie o functie **elibnod** care elibereaza zonele din memoria heap ocupate de nodul de tip NOD definit in exercitiul precedent.

O posibila solutie:

```
void elibnod(NOD *p)
/* elibereaza zonele din memoria heap ocupate de nodul spre care pointeaza p */
{
 free(p -> cuvant);
 free(p);
}
```

4) Sa se scrie o functie:

NOD *echivalenta(NOD *q, NOD *p)

pentru nodurile de tipul NOD, definit la punctul 2, care realizeaza urmatoarele:

- elibereaza zona de memorie ocupata de nodul spre care pointeaza **p**;
- incrementeaza valoarea componentei: **q -> frecventa**;
- returneaza valoarea lui **q**.

5) Sa se scrie functia:

void prelucrare(NOD *p)

care afiseaza datele care nu sunt de inlantuire dintr-un nod de tipul NOD, definit la punctul 2. Presupunand ca datele vor fi afisate continuu, introduce-ti o asteptare la 23 de afisari (randuri).
O posibila solutie:

```
void prelucrare(NOD *p) /*  afiseaza p -> cuvant si p -> frecventa */
{
```

```
 static int n = 0;
 printf("\nCuvantul: %ls are frecventa: %d", p -> cuvant, p -> frecventa);
 if((n+1)%23 == 0)
          {
           printf("\nActionati o tasta pentru a continua");
           getch();
          }
 n++;
}
```

6) Sa se scrie functia:

int criteriu(NOD *p1, NOD *p2)

care returneaza valorile:

-1 - daca p2->cuvant < p1->cuvant;
1 - daca p2->cuvant > p1->cuvant;
0 - daca p2->cuvant = p1->cuvant.

NOD este tipul descris la punctul 2.

O posibila solutie:

```
int criteriu(NOD *p1, NOD *p2)
          /* returneaza :
                    -1        - daca p2->cuvant < p1->cuvant;
                     1        - daca p2->cuvant > p1->cuvant;
                     0        - altfel.  */
{
 int i;
 if((i=stricmp(p2 -> cuvant, p1 -> cuvant)) < 0)
          return -1;
 else
          if(i>0)
                    return 1;
          else
                    return 0;
}
```

7) Sa se scrie o functie:

NOD *insnod()

care conform pasilor descrisi in prima parte a lucrarii insereaza un nod de tipul NOD, intr-un arbore binar. NOD este tipul descris la punctul 2.

O posibila solutie:

```
NOD *insnod()
/* - insereaza un nod in arborele binar spre a carui radacina pointeaza prad;
   - returneaza pointerul spre nodul inserat sau pointerul returnat de functia echivalenta daca nodul de inserat este echivalent cu unul deja aflat in arbore;
   - returneaza zero daca nu mai sunt date de incarcat sau la eroare.  */
{
 extern NOD *prad;
 int i;
 int n;
```

```
 NOD *p, *q;

 n = sizeof(NOD);
 if(((p = (NOD *)malloc(n)) != 0) && (incnod(p) == 1)) /* 1 */
          {
           p -> st = p -> dr = 0;

           if(prad == 0)  /* arbore vid */
                    {
                     prad = p; /* nodul curent devine radacina */
                     return p;
                    }

        /* se determina pozitia in arbore a nodului si se face inserarea daca este cazul */
           q = prad;
           for( ; ; )
                    {
                     if((i = criteriu(q,p)) < 0)
                              if(q -> st == 0)
                                       { /* se insereaza ca fiu stang */
                                        q -> st = p;
                                        return p;
                                       }
                              else /* se continua cautarea pozitiei in subarborele stang */
                                       {
                                        q = q -> st;
                                        continue;
                                       }
                     if(i > 0)
                              if(q -> dr == 0)
                                       { /* se insereaza ca fiu drept */
                                        q -> dr = p;
                                        return p;
                                       }
                              else /* se continua cautarea pozitiei in subarborele drept */
                                       {
                                        q = q -> dr;
                                        continue;
                                       }
                       /* nu se poate face inserarea si se apeleaza functia echivalenta */
                              return echivalenta(q,p);
                             }  /* sfarsit for */
          }  /* sfarsit if 1 */

 if(p == 0)
          {
           printf("\nMemorie insuficienta\n");
           getch();
           exit(1);
          }
 elibnod(p);
 return 0;
}
```

8) Sa se scrie functia:

void inord(NOD *p)

care parcurge un arbore binar in inordine.

O posibila solutie:

```
void inord(NOD *p)  /* parcurge un arbore binar in inordine */
{
 if(p != 0)
            {
             inord(p -> st);
             prelucrare(p);
             inord(p -> dr);
            }
}
```

9) Sa se scrie un program care citeste cuvintele dintr-un text si afiseaza numarul de aparitii al fiecarui cuvant din textul respectiv. Cuvantul se defineste ca o succesiune de litere mici si/sau mari. Textul se termina prin sfarsitul de fisier (CTRL+Z). Se va introduce modelul structurii NOD (a se vedea punctul 2) si se vor include toate functiile definite la punctele anterioare. Variabila globala **prad** se va declara ca pointer de tipul NOD. Programul construieste un arbore binar ale carei noduri au tipul NOD deja definit si il parcurge in inordine afisand fiecare cuvant si frecventa sa.

RASPUNSURI

4) ECHIV.CPP

```
NOD *echivalenta(NOD *q, NOD *p)
/* - elibereaza zona de memorie ocupate de nodul spre care pointeaza p;
   - incrementeaza pe q -> frecventa;
   - returneaza valoarea lui q.  */
{
 elibnod(p);
 q -> frecventa++;
 return q;
}
```

9) LAB_TP4.CPP

```
# include<stdio.h>
# include<stdlib.h>
# include<conio.h>
# include<string.h>

typedef struct nod
                        {
                         char *cuvant;
                         int frecventa;
                         struct nod *st;
                         struct nod *dr;
                        } NOD;

# include "citcuv.cpp"
# include "incnod.cpp"
# include "elibnod.cpp"
# include "echiv.cpp"
# include "prel.cpp"
# include "criter.cpp"
# include "insnod.cpp"
# include "inord.cpp"


NOD *prad;

void main()  /* citeste cuvintele dintr-un text si le afiseaza in ordine alfabetica impreuna cu frecventa de
aparitie a lor in text */
{
 /* construieste arbore binar */
 clrscr();
 prad = 0;
 while(insnod())
            ;
 /* parcurge arborele in inordine */
 inord(prad);
 getch();
}
```