7. Arbori.

Definiții şi generalități.

Un arbore este un model abstract al unei relații ierarhice. Un arbore constă din noduri aflate în relații de tip fiu-părinte.

Sistemul de fişiere şi mediile de programare sunt aplicații ce pot fi modelate folosind arbori. Structurile arborescente:

- sunt modele naturale pentru ierarhii de obiecte
- permit menținerea ordinii în colecții de date dinamice, în care se fac multe inserări şi ştergeri.
- sunt structuri de căutare rapidă

Un ***arbore liber*** **`T=(N,A)`** este un *graf conex aciclic*, în care **`N`** este o mulțime de noduri, iar **`A`** este o mulțime de arce care conectează perechi de noduri.

Arborele **`T=(N,A), n=|N|`** are următoarele proprietăți:

1. are **`n`** noduri şi **`n-1`** arce
2. nu are cicluri
3. între oricare pereche de noduri există o cale unică
4. ştergerea unui arc formează doi arbori
5. adăugarea unui arc crează un ciclu.

Un ***arbore cu rădăcină*** are următoarele proprietăți:

1. are un nod special numit *nod rădăcină*
2. fiecare nod (exceptând rădăcina) are un *părinte* (*predecesor* sau *tată*) unic
3. fiecare nod are 0 sau mai mulți subarbori T_1, T_2, ..., T_n nevizi, ai căror rădăcini sunt *fii* (*succesori*) ai nodului rădăcină.. Numărul de fii ai unui nod defineşte ***gradul*** *nodului*.
4. un nod fără succesori este un *nod frunză* (celelalte sunt *noduri interne*)
5. o *cale* între 2 noduri este o secvență de noduri şi arce:

   `cale(u,v)={u, (u,i), i, (i,j), j,…,k, (k,v), v}`

6. *lungimea căii* este numărul de arce de pe cale.
7. Un nod **`u`** este *strămoş* al unui nod **`v`** sau **`v`** este *descendent* al unui nod **`u`** dacă:

   `Pred(Pred(...Pred(v)...))=u`

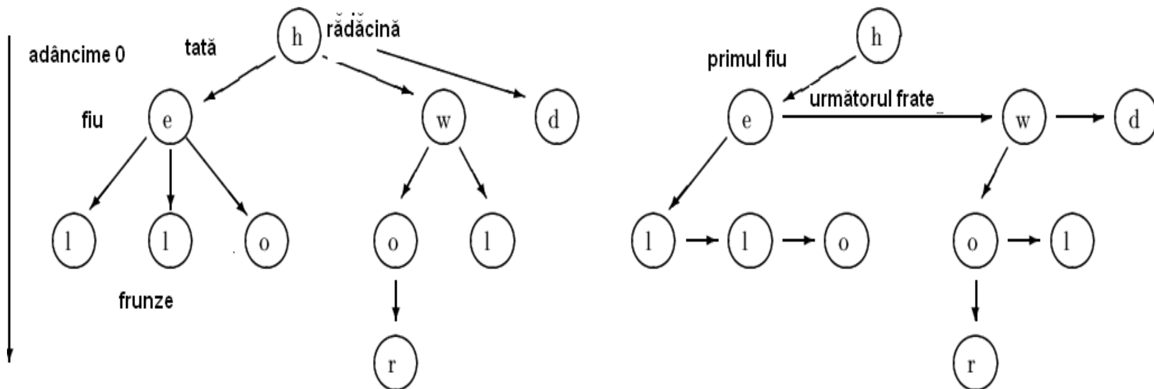
8. *Înălțimea unui nod* **`u`** este numărul de arce de pe cea mai lungă cale de la **u** la un nod frunză. Inălțimea unui nod se exprimă recursiv astfel:

$$h(n) = \begin{cases} 0 & \text{dacă } n \text{ este frunză} \\ 1 + \max_k(h(S_k(n))) & \end{cases}$$

   în care $S_k(n)$ este subarborele având ca rădăcină fiul c_k al nodului **`n`**.

9. *Înălțimea arborelui* este înălțimea rădăcinii, adică numărul de arce de pe cea mai lungă cale de la rădăcină la o frunză
10. *Adâncimea unui nod* este numărul de arce de la acel nod la rădăcină. Rădăcina are adâncimea 0, iar *un arbore vid are adâncimea* –1..

```
Adâncime(nod) = Adâncime(Pred(nod)) + 1
Adâncime(rădăcină) = 0
```



Intr-un arbore, pe nivelul 0 se află rădăcina, pe nivelul 1 – succesorii direcți ai rădăcinii, etc.

Succesorul unui nod este rădăcină a unui *subarbore*.

Un arbore în care nodurile pot avea orice grad este un ***arbore general*** sau *arbore multicăi*

Un arbore poate fi definit *recursiv* astfel:

1. un arbore este un singur nod – nodul rădăcină
2. un arbore este un nod rădăcină cu fiii c_1, c_2, ..., c_k şi aceste noduri sunt rădăcini de arbori.

Operațiile generale pe arbori multicăi se referă la:

- deplasarea în arbore
- traversarea arborelui
- căutarea, inserarea şi ştergerea de noduri

Într-un arbore binar gradul maxim al oricărui nod nu depăşeşte 2.

Un arbore binar se defineşte recursiv ca:

- un arbore vid
- un arbore cu un nod rădăcină cu doi fii: stâng şi drept, care sunt rădăcini de arbori binari.

7.1. Arbori Binari.

Operații specifice TAD Arbore Binar.

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

`Arb  AB_New();` – crează un arbore binar vid
`int  AB_Empty(Arb a);` – test arbore vid
`int  AB_Size(Arb a);` – numărul de noduri al arborelui **a**
`Arb  AB_Root(Arb a);` – dă rădăcina nodului **a**
`void *AB_GetKey(Arb a);` – dă cheia din nodul **a**
`void AB_SetKey(Arb a, void *k);` – face adresa cheii nodului **a** egală cu **k**
`int  AB_Leaf(Arb a);` – **1** dacă nodul **a** este frunză, **0** altfel
`Arb  AB_GetPred(Arb a);` – dă nodul tată al nodului **a** sau **NULL**
`void AB_SetPred(Arb a, Arb p, int dir);`–face nodul **p** tată al nodului **a**.
Nodul predecesor **p** va avea nodul **a** subarbore stâng sau drept,după cum **dir** este -1 sau 1
`Arb  AB_GetSS(Arb a);` – dă rădăcina subarborelui stâng a lui **a**
`void AB_SetSS(Arb a, Arb s);` – face **s** subarbore stâng a lui **a**
`Arb  AB_GetSD (Arb a);` – dă rădăcina subarborelui drept a lui **a**

void AB_SetSD(Arb a, Arb d); - face **d** subarbore drept a lui **a**
Arb AB_CrNod(void *ch, Arb s, Arb d, Arb p, int dir);-crează un nod cu cheie, subarbori şi părinte specificaţi; nodul creat este subarbore stâng sau drept a lui **p** după cum **dir** este -1 sau 1.
Arb AB_CrArb(Arb r, Arb s, Arb d); -crează un arbore dintr-un nod rădăcină şi doi subarbori specificaţi
int AB_Height(Arb a); **-** dă înălţimea arborelui **a**
int AB_Depth(Arb n); **-** dă adâncimea nodului **n**

Traversarea arborilor binari şi aplicaţii ale traversărilor.

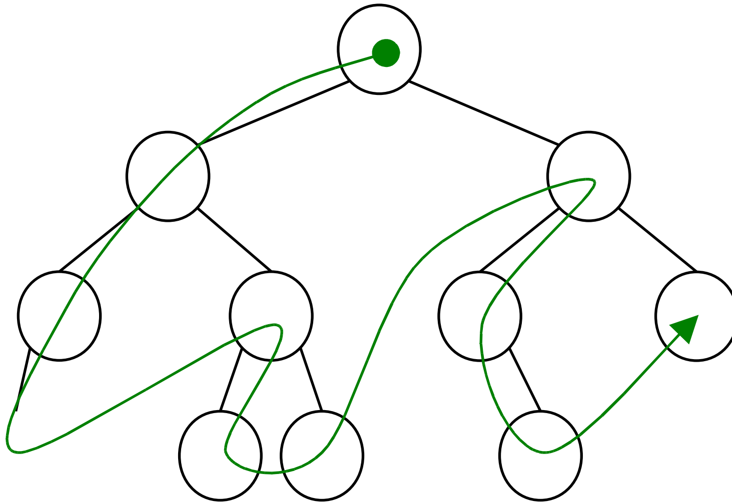
Traversarea , *parcurgerea* sau *vizitarea* unui arbore binar reprezintă o modalitate sistematică de enumerare a nodurilor arborelui, în scopul efectuării unei prelucrări asupra nodurilor.
Se pot defini următoarele traversări de arbori binari:

- traversare în preordine – **RSD** : rădăcină – subarbore stâng – subarbore drept
- traversare în inordine – **SRD** : subarbore stâng – rădăcină – subarbore drept
- traversare în postordine – **SDR** : subarbore stâng – subarbore stâng – rădăcină
- traversare în inordine inversă – **DRS** : subarbore drept – rădăcină – subarbore stâng
- traversare în lăţime

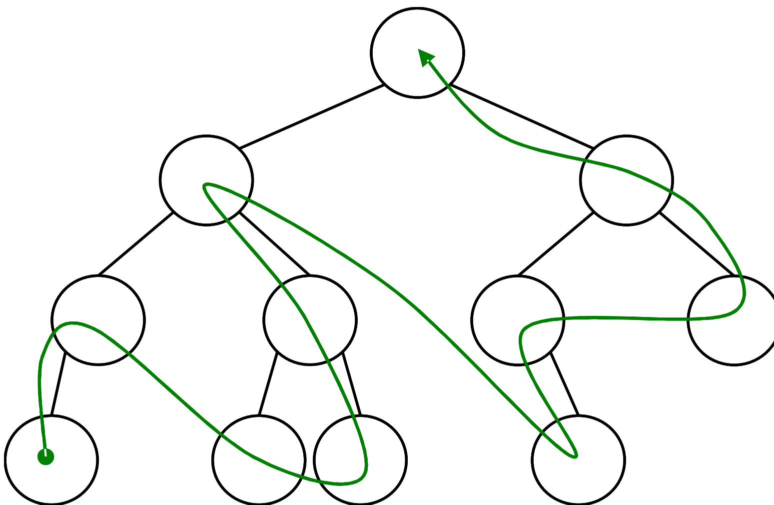
```
void PreOrd(Arb a, void (*Vizitare)(void*)){
   if(!AB_Empty(a)){
      Vizitare(AB_GetKey(a));
      PreOrd(AB_GetSS(a), Vizitare);
      PreOrd(AB_GetSD(a), Vizitare);
   }
}
```

O traversare nerecursivă în preordine utilizează o stivă:

```
void PreOrd(Arb a){
   Stiva S=S_New ();
   Push(S, a);
   while(!S_Empty(S)){
      Arb n=Pop(S);
      Vizitare(AB_GetKey(n));
      if(AB_GetSS(n)) Push(S, AB_GetSS(n));
      if(AB_GetSD(n)) Push(S, AB_GetSD(n));
  }
}
```



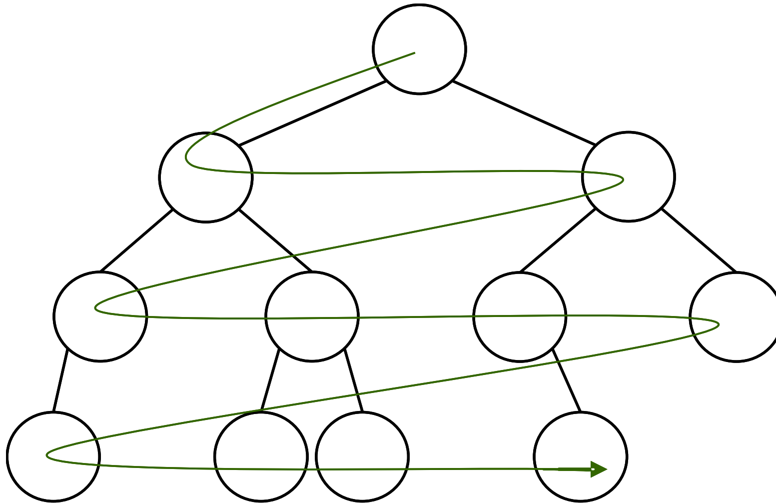
```
void InOrd(Arb a){
   if(!AB_Empty(a)){
      InOrd(AB_GetSS(a));
      Vizitare(AB_GetKey(a));
      InOrd(AB_GetSD(a));
   }
}
void PostOrd(Arb a){
   if(!AB_Empty(a)){
      PostOrd(AB_GetSS(a));
      PostOrd(AB_GetSD(a));
      Vizitare(AB_GetKey(a));
   }
}
```



Traversarea în lățime a unui arbore binar.

Traversarea în lățime (sau pe niveluri) a unui arbore binar presupune folosirea unei cozi în care se pun adresele nodurilor. Cât timp coada nu este vidă, se scoate un nod, se prelucrează şi i se pun succesorii în coadă.

```
void travLat(Arb a, void (*Vizit)(Arb )){
  Coada Q = Q_New ();
  Arb p;
  Enq(Q, a);
  while(!Q_Empty(Q)){
    p = Deq(Q);
    Vizit(AB_GetKey(p));
    if(AB_GetSS(p))
      Enq(Q, AB_GetSS(p));
    if(AB_GetSD(p))
      Enq(Q, AB_GetSD(p));
  }
}
```



Exemplele se încadrează în traversarea în postordine:

1. *ştergerea unui arbore binar*

```
void AB_Delete(Arb *a){
  if(!AB_Empty(*a)){
    AB_Delete(&AB_GetSS(*a));
    AB_Delete(&AB_GetSD(*a));
    free(*a);
    *a = NULL;
  }
}
```

2. *copierea un unui arbore binar*

```
Arb copie (Arb a){
  if(AB_Empty(a)) return NULL;
  Arb b = AB_CrNod(AB_GetKey(a),NULL,NULL,NULL,0);
```

```
    Arb s = copie(AB_GetSS(a));
    Arb d = copie(AB_GetSD(a));
    return AB_CrArb(b,s,d);
  }
```

3. *Afişarea unui arbore binar..*

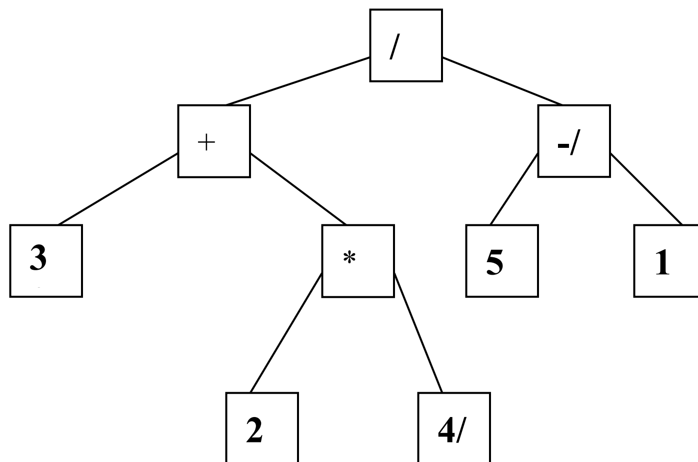
O traversare mai puţin obişnuită – traversarea în inordine inversată DRS se foloseşte pentru afişarea arborilor binari. Dacă rotim un arbore binar cu 90^0, rădăcina va apare în stânga, nodurile din partea dreaptă vor apare primele, aşa că trebuie efectuată o parcurgere DRS. Fiecare nivel din arbore va fi indentat cu 5 spaţii libere.

```
void indent(int niv){
  for(int j=0; j<niv; j++)
    printf( "     ");
}

void afRot(Arb a, int niv){
  if(a){
    afRot(AB_GetSD(a), niv+1);
    indent(niv);
    printf(" %4d\n", *(int*)AB_GetKey(a));
    afRot(AB_GetSS(a), niv+1);
  }
}
```

4. *Evaluarea unui arbore expresie.*

O expresie aritmetică poate fi reprezentată printr-un arbore, având termenii ca noduri frunze şi operatorii ca noduri interne. In reprezentarea expresiei nu sunt necesare paranteze.De exemplu expresia
`(3 + 2 * 4) / (5 – 1)` se reprezintă prin arborele:



Pentru evaluarea arborelui expresie se evaluează mai întâi subarborii stâng şi drept şi apoi se aplică operatorul. Nodurile frunze conţin termeni. Traversarea arborelui expresie se face în postordine.Cheile sunt pointeri la şiruri de caractere.

```
double eval(Arb a){
  if(AB_Leaf(a)) return atod(AB_GetKey(a));
```

```
  char op = *(AB_GetKey(a));
  double x = eval(AB_GetSS(a));
  double y = eval(AB_GetSD(a));
  switch(op){
    case '+': return x+y;
    case '-': return x-y;
    case '*': return x*y;
    case '/': return x/y;
  }
}
```

5. *Afişarea unui arbore sub formă parantezată*.

Pentru afişarea unui arbore sub formă parantezată se verifică nodul curent: dacă este rădăcină sau frunză – se afişează. In caz contrar, se afişează "(", apoi subarborele stâng, ",", subarborele drept şi ")".

```
void printArbPar(Arb a){
  if(AB_Root(a) || AB_Leaf(a))
    printf("%4d", *(int*)AB_GetKey(a));
  else
    { printf("(");
      printArbPar(AB_GetSS(a));
      printf(",");
      printArbPar(AB_GetSD(a));
      printf(")");
    }
}
```

Afişarea unui arbore expresie se face astfel:

- pentru o frunză (termen) se face afişare
- pentru un nod intern (operator) se afişează:
  - (
  - subarborele stâng
  - operator
  - subarborele drept
  -)

Fiind o structură definită recursiv, operațiile uzuale cu arbori binari, pot fi de asemeni reprezentate în mod recursiv.
Căutarea unei chei în arborele binar presupune căutarea în nodul rădăcină, şi dacă nu se găseşte se continuă căutarea în subarborele stâng şi apoi dacă este cazul în subarborele drept. Algoritmul nu poate fi considerat o traversare, deoarece nu sunt vizitate toate cheile, oprindu-ne la găsirea cheii căutate.

6. *căutarea unei valori într-un arbore binar*

```
Arb Search(Arb a, void *x, PFC comp){
  if(AB_Empty(a)) return NULL;
  if(comp(AB_GetKey(a), x)==0) return a;
  Arb p = Search(AB_GetSS(a), x, comp);
  if(p) return p;
  return Search(AB_GetSD(a), x, comp);
}
```

Implementarea arborilor binari.

Un arbore binar poate fi reprezentat printr-un vector sau folosind pointeri.

5.1. Arbori binari implementați cu pointeri.

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;
```

Utilizatorul, care nu cunoaşte detaliile structurii nod, va folosi funcțiile precizate mai sus

```
Arb  AB_New (){return NULL;}
int  AB_Empty(Arb a){return a==NULL;}
```

Numărarea nodurilor se face recursiv:

```
int  AB_Size(Arb a){
  if(!a) return 0;
  return AB_Size(a->stg)+AB_Size(a->dr)+1;
}

void *AB_GetKey(Arb a){
  assert(a);
  return a->key;
}

void AB_SetKey(Arb a, void *k){
  assert(a);
  a->key = k;
}

int  AB_Leaf(Arb a){
  assert(a);
  return (a->st==0 && a->dr==0);
}

Arb  AB_Root(Arb a){
  assert(a);
  while(a->pred)
     a = a->pred;
  return a;
}

Arb AB_GetPred(Arb n){
  assert(n);
  return n->pred;
```

```
}
Arb  AB_GetSS(Arb a){
  assert(a);
  return a->st;
}
Arb  AB_GetSD(Arb a){
  assert(a);
  return a->dr;
}
void AB_SetSS(Arb a, Arb s){
  assert(a);
  a->st = s;
  if(s)
    s->pred = a;
}
void AB_SetSD(Arb a, Arb d){
  assert(!AB_Empty(a));
  a->dr = d;
  if(d)
    d->pred = a;
}
void AB_SetPred(Arb a, Arb p, int dir){
   assert(!AB_Empty(a));
   a->pred = p;
   if(dir==-1)
      p->st = a;
   else
      if(dir==1)
         p->dr = a;
}
Arb AB_CrNod(void *ch, Arb s, Arb d, Arb p, int dir){
   Arb r = (Arb)malloc(sizeof(struct arb));
   r->key = ch;
   r->st = s;
   r->dr = d;
   r->pred = p;
   if(p)
      switch(dir){
         case -1: p->st = r; break;
         case  1: p->dr = r; break;
         case  0: ;
      }
   return r;
}
Arb  AB_CrArb(Arb r, Arb s, Arb d){
  assert(r);
  AB_SetSS(r, s);
  AB_SetSD(r, d);
```

```
  return r;
}
int AB_Height(Arb a){
   int hs, hd;
   if(!a) return -1;
   hs = AB_Height(a->st);
   hd = AB_Height(a->dr);
   return 1 + (hs > hd? hs: hd);
}
int  AB_Depth(Arb n){
  if(!n) return -1;
  return 1 + AB_Depth(n->pred);
}
```

5.2. Arbori binari implementați cu pointeri simulați (cursori).

Pointerii sunt simulați prin întregi reprezentând poziții ale valorilor în tablouri. Am păstrat numai pointerii generici la chei pentru a permite reprezentarea independentă de tip
Alocarea de memorie pentru tablouri se va face, din considerente de eficiență, o singură dată, prin constructor.

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

5.3. Arbori binari reprezentați prin vectori.

Acest mod de reprezentare este specific arborilor binari compleți şi va fi prezentat ulterior.

Probleme propuse.

1. Scrieți o funcție nerecursivă care care traversează un arbore binar în preordine afişând în cursul traversării cheile caractere, în două variante

a) folosind o stivă
b) fără a folosi stivă.

Indicații:
a) Se coboară pe subarborele stîng, punând subarborele drept în stivă (dacă există). La întâlnirea unui subarbore stâng vid, se scoate subarborele drept din stivă.

b) Se coboară, pornind din rădăcină pe fiul stâng până în frunză, afişând cheile traversate. Se revine trecându-se pe frate drept sau tată în absența fratelui drept. Traversarea se consideră încheiată când se revine în rădăcină din subarborele drept, sau nu există subarbore drept.
Se pot folosi funcțiile interfeței arborelui binar.

2. Scrieți o funcție care numară nodurile frunze de pe un nivel dat al unui arbore binar.

3. Scrieţi o funcţie având ca parametru un arbore expresie, care crează un şir de caractere continând reprezentarea expresiei astfel:
- pentru un nod frunză se pune în şirul de caractere reprezentarea externă a termenului
- pentru un nod intern se pun în expresie "(", `ss`, operator, `sd`, ")".

4. a) Pentru un arbore binar, având cheile din noduri caractere, traversările în inordine şi preordine sunt respectiv: "**ACZGYXBTDHMF**" şi "**AXYCZGBDTFHM**"
Reconstituiţi pe baza lor arborele binar.
b) Scrieţi o funcţie recursivă, având ca parametri şirurile de caractere reprezentând cele două traversări, funcţie care întoarce adresa nodului rădăcină al arborelui reconstituit

5. a) Pentru un arbore binar, având cheile din noduri caractere, traversările în inordine şi postordine sunt respectiv: "**ATZBYCGEHDFX**" şi "**ZTYBGHEFDCAX**"
Reconstituiţi pe baza lor arborele binar.
b) Scrieţi o funcţie recursivă, având ca parametri şirurile de caractere reprezentând cele două traversări, funcţie care întoarce adresa nodului rădăcină al arborelui reconstituit

6. a) Pentru un arbore binar, având cheile din noduri caractere, traversările în inordine şi pe niveluri (în lăţime) sunt respectiv:"**YEZAMCXTFBDG**" şi "**MAXYCBZTDEFG**"
Reconstituiţi pe baza lor arborele binar.
b) Scrieţi o funcţie recursivă, având ca parametri şirurile de caractere reprezentând cele două traversări, funcţie care întoarce adresa nodului rădăcină al arborelui reconstituit

7. Pentru reconstituirea unui arbore binar având chei caractere, folosind numai traversarea în preordine mai sunt necesare anumite informaţii. Astfel fiecărui nod din traversare i se asociază un caracter cu următoarele semnificaţii:
'F' – nodul este frunză
'L' – nodul are numai fiu stânga
'R' – n odul are numai fiu dreapta
'2' – nodul are ambii succesori
a) Dându-se şirurile de caractere "**MAYZEXCBTFDG**" şi"**2LRLF2F2RFRF**" reconstituiţi arborele
b) Scrieţi o funcţie având ca parametri două şiruri de caractere conţinând traversarea în preordine şi proprietăţile nodurilor traversate, funcţie care întoarce un pointer la rădăcina arborelui reconstituit

8. Scrieţi o funcţie nerecursivă de traversare a unui arbore binar folosind o stivă:
a) în postordine
b) în preordine

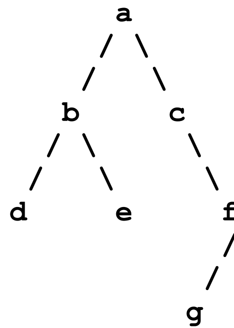
9. Definiţi o funcţie nerecursivă care numără frunzele unui arbore binar folosind o stivă.

10. Scrieţi o funcţie nerecursivă care numără nodurile frunze (noduri interne având ca fii noduri externe) ale unui arbore binar, folosind o stivă.

Indicaţie: Se numără nodurile frunze pe stânga, coborând în subarborele stâng şi punând subarborele drept în stivă. La întâlnirea unui subarbore stâng vid se scoate subarborele drept din stivă.

10. Un arbore binar are chei caractere. Pentru acesta se dau două şiruri de caractere: traversările în preordine şi inordine.

De exemplu, pentru arborele

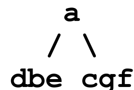


traversarea în preordine este: **`abdecfg`** , iar cea în inordine este **`dbeacgf`** .
Traversarea în postordine a arborelui este: **`debgfca`**

a) Dându-se traversările în preordine: **`vaetilnrq`** şi în inordine **`ateivnqrl`**, daţi traversarea în postordine.

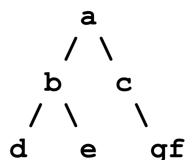
b) Scrieţi o funcţie având ca parametri două şiruri de caractere (traversările în preordine şi inordine), care calculează ca rezultat un al treilea şir de caractere (traversarea în postordine.

Indicaţie: Pe baza traversărilor se reconstituie arborele, în felul următor: se parcurge traversarea în preordine de la stânga la dreapta. Caracterul curent (**a**) este rădăcina unui arbore. Cei doi subarbori ai acestuia se găsesc în stânga şi dreapta caracterului, în traversarea în inordine **`dbeacgf`**.



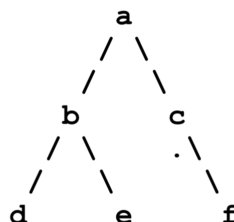
Un subarbore este precizat prin cheile cuprinse între doi indici **i** şi **j** în traversarea în inordine. Pentru fiecare subarbore, se determină, în acelaşi mod (recursiv) rădăcina şi subarborii. Se iasă din recursivitate la întâlnirea unui nod vid. Arborele reconstituit este traversat în postordine.

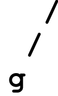
În exemplul de mai sus, rădăcina arborelui **a** este preluată din **`preord[0];`** ea se regăseşte în **`inord`** în poziţia 3 şi delimitează doi subarbori având chei în intervalele **`0:2`**, respectiv **`4:6`**. Următoarea cheie **`b`** este găsită în poziţia **1** şi delimitează subarborii cu chei în **`0:0`** şi **`2:2`** .



Cheia următoare **d** este găsită în poziţia **0** şi nu are subarbori; la fel cheia **e**, aflată în poziţia **2** nu are subarbori, etc.

11. Un arbore binar are chei caractere. Pentru acesta se dau două şiruri de caractere: traversările în postordine şi inordine.
De exemplu, pentru arborele



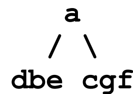


traversarea în postordine este: : **debgfca**, iar cea în inordine este **dbeacgf** .
Traversarea în preordine a arborelui este **abdecfg**

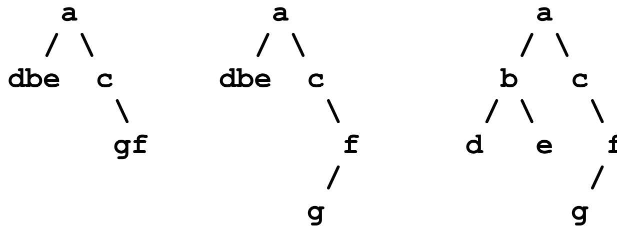
a) Dându-se traversările în postordine:**otelnrauc** şi în inordine: **toaenlrcu**, daţi traversarea în preordine.

b) Scrieţi o funcţie având ca parametri două şiruri de caractere (traversările în postordine şi inordine), care calculează ca rezultat un al treilea şir de caractere (traversarea în preordine.

Indicaţie: Pe baza traversărilor se reconstituie arborele, în felul următor: se parcurge traversarea în postordine de la dreapta la stânga. Caracterul curent (**a**) este rădăcina unui arbore. Cei doi subarbori ai acestuia se găsesc în stânga şi dreapta caracterului, în traversarea în inordine **dbeacgf.**



Pentru fiecare subarbore, se determină, în acelaşi mod (recursiv) rădăcina şi subarborii. Se iasă din recursivitate la întâlnirea unui nod frunză.



Arborele reconstituit este traversat în postordine.

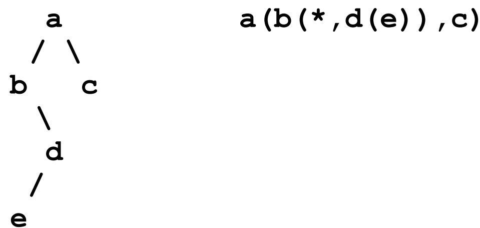
În exemplul de mai sus, rădăcina arborelui **a** este preluată din **preord[n-1];** ea se regăseşte în **inord** în poziţia 3 şi delimitează doi subarbori având chei în intervalele **0:2**, respectiv **4:6**. Următoarea cheie **c** este găsită în poziţia **4** şi delimitează numai un subarbore drept cu chei în **5:6** Cheia următoare **f** este găsită în poziţia **6** şi are numai subarbore stâng cu chei în **5:5**; apoi cheia **g**, aflată în poziţia **5** nu are subarbori, etc.

Un arbore binar este reprezentat sub forma **R(S D)**, în care **R** este cheia din rădăcină (un caracter), iar **S** şi **D** sunt subarborii stâng şi drept, definiţi recursiv ca arbori binari.
Dacă subarborele stâng lipseşte, arborele va fi reprezentat ca **R(* D)**, iar dacă subarborele drept lipseşte, reprezentarea va fi **R(S)**.
Scrieţi o funcţie recursivă, având ca parametru un arbore binar, care întoarce ca rezultat un şir de caractere reprezentând expresia cu paranteze corespunzătoare arborelui.

Un arbore binar este reprezentat sub forma **R(S D)**, în care **R** este cheia din rădăcină (un caracter), iar S şi D sunt subarborii stâng şi drept, definiţi recursiv ca arbori binari.
Dacă subarborele stâng lipseşte, arborele va fi reprezentat ca **R(* D)**, iar dacă subarborele drept lipseşte, reprezentarea va fi **R(S)**.
Scrieţi o funcţie recursivă. având ca parametru un şir de caractere descriind un arbore binar, funcţie care crează arborele binar corespunzător şi întoarce un pointer la rădăcina acestuia.

Funcție pentru determinarea înălțimii unui arbore binar, ca înălțime maximă dintre cei doi subarbori, plus unu.

Un arbore binar poate fi reprezentat sub forma **`R(S,D)`**, în care **`R`** desemneaza cheia din rădăcină, reprezentată printr-o literă, iar **`S`** şi **`D`** sunt subarborii stâng si drept.
Daca subarborele stâng lipseşte, arborele va fi reprezentat sub forma **`R(*,D)`**, iar daca subarborele drept lipseste, arborele binar va fi reprezentat ca **`R(S)`**.
Definiți o funcție recursiva, **`creere_exp(Arb a, char* expr)`**, având ca parametri un arbore binar şi un şir de caractere, care crează structura arborelui sub forma unei expresii cu paranteze si o depune în parametrul **`expr`**. De exemplu pentru arborele binar dat mai jos, funcția va crea în al doilea parametru expresia:



Operația inversa, de creere în memorie a unui arbore binar pe baza unei expresii cu paranteze se exprima printr-o strategie divide et impera astfel:

```
void creare_arb(nod*& a, char *expr)
{ char *es, *ed;
  if (expr)
    if (expr[0] != '*') {
        legare(a, expr[0]);
        separa(expr, es, ed);
        creare_arb(a->st, es);
        creare_arb(a->dr, ed);
    }
}
```

în care:
- funcția "legare" ataseaza la un pointer dat ca prim argument un nod avand drept cheie al doilea parametru
- funcția "separa" creeaza din arborele descris prin primul argument sir de caractere reprezentarile prin formule (siruri de caractere) ale celor doi subarbori (stang si drept).
Scrieți funcțiile "**`legare()`**" si "**`separa()`**", care implementeaza funcția de creere a arborelui binar.

Funcție pentru afişarea arborelui sub forma unei expresii cu paranteze (pornind de la explorarea prefixată).
De exemplu, pentru arborele creat cu prima funcție şi cheile "**`msdpetbh`**", prin aplicarea acestei funcții se obține expresia: **`m(d(b,e( ,h)),s(p,t))`**

Funcție pentru desenarea structurii unui arbore binar intr-o folosind o traversare in latime.
Exemplu:

b **e** **p** **t**
h

Funcție pentru eliminarea unui nod cu valoare dată dintr-un arbore binar de căutare.

Scrieți o funcție `main()` în care creați, pe baza unui şir de caractere citit, un arbore binar de căutare, îl afişați, determinați expresia cu paranteze şi o afişați, determinați şi afişați numărul de noduri şi înălțimea arborelui şi apoi ştergeți câte un nod, afişând de fiecare dată arborele rămas.

Pentru reconstituirea unui arbore binar având chei caractere, folosind numai traversarea în preordine mai sunt necesare informații asupra tipului nodurilor. Astfel fiecărui nod din traversare i se asociază un caracter cu următoarele semnificații:
'F' – nodul este frunză
'L' – nodul are numai fiu stânga
'R' – nodul are numai fiu dreapta
'2' – nodul are ambii succesori

a) Dându-se şirurile de caractere "**MAYZEXCBTFDG**" şi"**2LRLF2F2RFRF**" reconstituiți arborele
b) Scrieți o funcție **`Arb RecAB(char *ch, char *tip)`**, având ca parametri două şiruri de caracteren conținând cheile caractere, la traversarea în preordine şi proprietățile nodurilor traversate, funcție care întoarce un pointer la rădăcina arborelui reconstituit.
Indicație: Se alocă memorie pentru nodul rădăcină, care devine nod curent şi se pune cheia în acesta.
Se repetă pentru toate celelalte noduri:
-se verifică numărul de succesori ai nodului curent:

- dacă sunt 2 se alocă memorie pentru două noduri care se leagă la arbore şi se pune adresa fiului drept în stivă. Nodul stâng devine nod curent.
- dacă există un singur succesor (stâng sau drept), se face celălalt NULLse alocă memorie pentru nod, care devine nod curent
- dacă este frunză, se scoate din stivă o adresă de nod, care devine nod curen

- se pune cheia în nodul curent

Un arbore binar are drept chei numere întregi. Definiți o funcție cu semnătura:
`Arb suma(Arb a);` care adună la cheia din fiecare nod suma cheilor din nodurile descendente. Funcția întoarce un arbore cu aceeaşi structură, dar cu cheile modificate.

.Scrieți o funcție **`void Print(Arb a, int h);`** care afişează cheile caractere ale arborelui binar **a**, astfel:

-cheia din rădăcină este afişată pe o linie, fiind precedată de $\frac{h}{2}$ spații

-cheia dintr-un alt nod este precedată de:

$L - \frac{h}{2^{d+1}}$ spații, dacă nodul este fiu stânga al părintelui, sau

$L + \frac{h}{2^{d+1}}$ spații, dacă nodul este fiu dreapta al părintelui.

L este indentarea (poziționarea) nodului părinte, iar **d** adâncimea nodului curent.

Indicaţie: Nodurile se vor vizita pe niveluri (deoarece nodurile de pe un nivel se vor afişa pe aceeaşi linie), ceea ce impune folosirea unei cozi. La afişarea cheii dimtr-un nod se pune poziţia în nod, pentru a fi folosită de nodurile fii. Sugerăm ca informaţia din nod să fie de tipul:

```
typedef struct{
      char cheie;
      int  poz;
} info;
```

Arbori generali (multicăi).

Interfaţa TAD Arbore General.

Introducem tipul **Arb** pentru referirea la un nod oarecare din arbore, inclusiv nodul rădăcină. Precizarea tipului depinde de modul de reprezentare al arborelui.

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

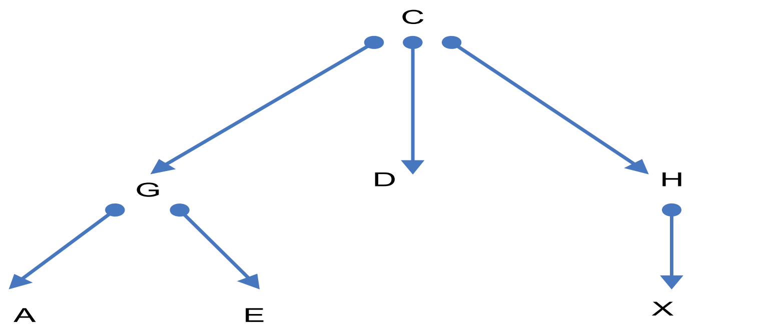
```
Arb A_New (Arb r);              // crează un arbore cu rădăcina r
void A_Delete(Arb *a);          //distruge arborele  a
int A_Size(Arb a);              // determină numărul de noduri al arborelui
Arb A_Root(Arb a);              // dă rădăcina arborelui la care aparţine nodul a
int A_Depth(Arb a, Arb n);      // adâncimea vârfului n din arborele a
int A_Height(Arb a);            //înălţime arbore  a
```

Operaţii specifice nodurilor

```
Arb  A_Pred(Arb n);             // dă nodul părinte a nodului n (NULL  pentru  rădăcină)
Arb  A_PrimFiu(Arb n);          //primul fiu al nodului  n
Arb  A_FrateUrm(Arb n);         //fratele următor al nodului n
Arb  A_FratePrec(Arb n);        //fratele precedent al nodului  n
int  A_Leaf(Arb n);             // 1 dacă  n  este frunză
void A_InsPrimFiu(Arb a, Arb n);  //inserează arborele n ca prim fiu a lui a
void A_InsFrateUrm(Arb a, Arb n); //inserează arborele n ca frate următor lui a
void A_RemovePrimFiu(Arb a);      //şterge primul fiu a lui a
void A_RemoveFrateUrm(Arb a);     //şterge frate următor a lui a
void A_Modif(Arb a, void *ch);    //schimbă cheia din nodul  a
```

3. Implementarea arborilor generali.

Sunt două alternative de reprezentare: cu tablouri şi cu pointeri



:

prin tabel de cursori (pointeri simulaţi) la predecesori

Reprezentare se face prin 2 tablouri:

- un tablou de chei asociate celor **n** noduri: **void **ch;**
- un tablou cu pozițiile predecesorilor nodurilor: **int *pred;**

```
struct arb{
   int  n;
   void **ch;
   int  *pred;
};
```

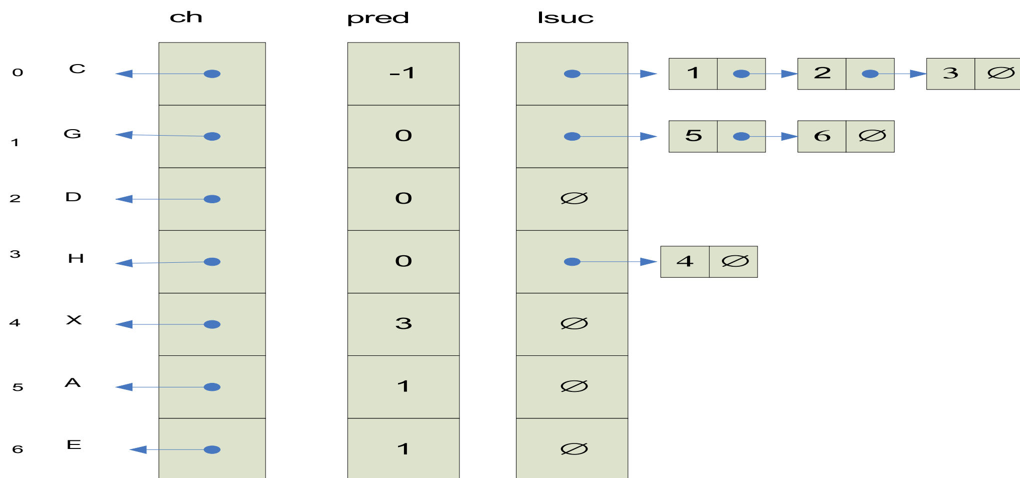
În acest mod unele dintre operații se fac mai puțin eficient (deplasarea prin arbore şi adăugarea de noi noduri se fac simplu, pe când celelalte operații sunt mai complicate).
Reprezentarea este folosită pentru *colecții de mulțimi disjuncte* şi va fi prezentată în acel context.

cu tablouri, prin liste de adiacențe

Elementul `lsuc[k]` din tabloul de adiacențe `lsuc` este un pointer la lista de succesori ai nodului `k.`.Pentru a permite şi accesul în sus trebuie adăugat un tabel de predecesori.

```
struct nodl{
   int crs;
   struct nodl *next;
};
```

```
struct arb{
   int    n;
   void   **ch;
   struct nodl **lsuc;
   int    *pred;
};
```



prin tablourile PrimFiu şi UrmFrate

Pentru a permite deplasare rapidă în sus în arbore trebuie adăugat şi un tabel de predecesori.

```
struct arb{
   void **Chei;
   int *PrimFiu;
   int *UrmFrate;
   int *Pred;
};
```

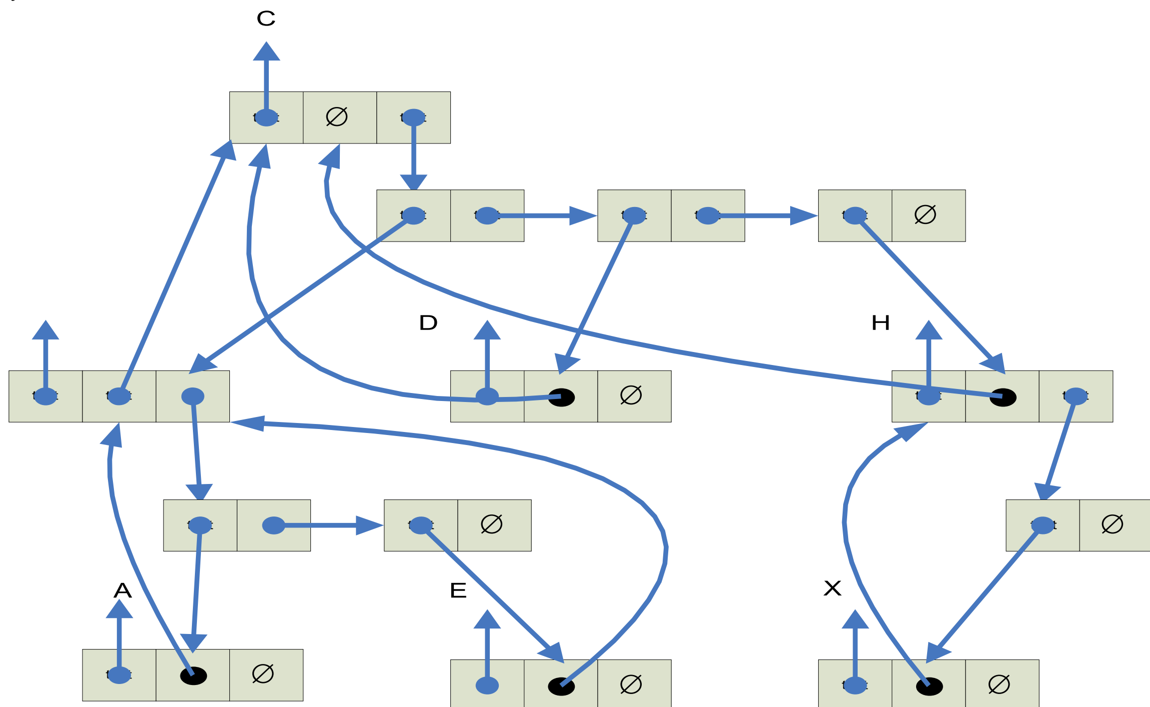
	ch	pred	prim_fiu	urm_frate
0	C	-1	1	-1
1	G	0	5	2
2	D	0	-1	3
3	H	0	4	-1
4	X	3	-1	-1
5	A	1	-1	6
6	E	1	-1	-1

4.cu pointeri, cu listă de succesori și pointer la predecesor

Vom folosi două structuri, una definind nodurile, cealaltă lista de adiacențe asociată unui nod.

```
struct ls;
struct nod{
   void *ch;
   struct nod *pred;
   struct ls  *primfiu:
};

struct ls{
   struct nod *fiu;
   struct ls  *urmfrate;
};
```



4. Traversarea arborilor generali.

Traversarea unui arbore presupune vizitarea fiecărui nod, o singură dată, într-o anumită ordine.

Există două tipuri de traversări:

- traversare în adâncime
- traversare în lățime

Traversarea în adâncime încearcă, poirnind din rădăcină, vizitarea nodului cel mai îndepărtat, care trebuie să fie fiu al unui nod deja vizitat.

Traversarea în adîncime se exprimă printr-un algoritm recursiv sau foloseşte o stivă.

Un algoritm generic de traversare în adâncime are forma:

```
void DFS(nod T)
   PreVizita(T)
   for(fiecare fiu c a lui T)
      DFS(c)
   PostVizita(T);
```

Dacă se foloseşte **`Previzita()`** avem o traversare în preordine, în timp ce folosirea **`PostVizita()`** defineşte o traversare în postordine.

Traversarea în lățime vizitează mai întâi nodurile cele mai apropiate, care nu au fost încă vizitate.

Traversarea în lățime (sau pe niveluri) se exprimă printr-un algoritm nerecursiv care foloseşte o coadă.

```
void BFS(nod T)
   Coada Q=Q_New ()
   Enq(Q, T)
   while(!Q_Empty(Q))
      nod u=Deq(Q)
      Vizitare(u)
      for(fiecare fiu c a lui u)
          Enq(Q, c)
```