**ARBORI CU RĂDĂCINĂ**

**1. Definirea arborilor cu rădăcină**

**Definiţie:** Se numeşte arbore un graf conex şi fără cicluri.

 În continuare se vor lua în discuţie arborii cu rădăcină. În cazul arborilor cu rădăcină se pune în evidenţă un vârf special al său numit rădăcină. Alegerea rădăcinii duce la aşezarea arborelui pe nivele astfel:

* se aşează răcina pe nivelul **1** (acest nivel se notează în multe lucrări de specialitate cu 0)
* pe fiecare nivel **k** (**k>1**) se plasează acele vârfuri pentru care lungimea lanţurilor care le leagă de rădăcină este **k-1**
* se trasează muchiile arborelui

Putem da şi o ***definiţie recursivă*** a unui arbore cu rădăcină: Se numeşte arborescenţă un arbore care are un vârf special numit **rădăcină** iar celelalte noduri pot fi repartizate în m mulţimi disjuncte **X1, X2,….,Xm, m≥0**, astfel încât în fiecare din aceste mulţimi există un nod adiacent cu rădăcina, iar subgrafurile generate de acestea sunt la rândul lor arborescenţe.

În particular, o arborescenţă cu un singur nod este formată doar din nodul rădăcină. Dacă ordinea relativă a arborescenţelor generate de mulţimile **X1, X2,….,Xm**din definţie are importanţă, arborescenţa se numeşte **arbore ordonat**.

În reprezentarea grafică a unei arborescenţe nodurile se desenează pe nivele, astfel: rădăcina se află pe primul nivel, vârfurile adiacente cu rădăcina pe următorul nivel, etc.

Nodurile adiacente cu rădăcina se numesc **descendenţii rădăcinii**. Conceptul se aplică analog pentru nodurile de pe un alt nivel. Descendenţii aceluiaşi nod se numesc **fraţi**. Dacă nodul **x** este descendentul unui nod **y**, îl numim pe acesta din urmă **părintele** nodului **x**.

Se numeşte **înălţimea unui arbore** diferenţa dintre nivelul maxim din arbore şi nivelul minim (nivelul rădăcinii). Pentru arborele de mai sus, înălţimea este 4-1=3.

**Observaţie:** Într-un arbore orice nod este legat de rădăcină printr-un lanţ unic de lungime minimă (corespunzător parcurgerii **BF** de la grafuri).

**2. Reprezentarea arborilor oarecare**

Nodurile unui arbore oarecare pot fi reprezentate sub forma unei zone de memorie alocată dinamic sau sub forma unui tablou de date alocate static. Fie următorul arbore:

**Reprezentarea** unui arbore oarecare se poate face astfel:

1. **Scrierea parantezată**: Fiind dat un arbore cu vârfurile etichetate (numere sau caractere), definim scrierea lui parantezată ca fiind scrierea parantezată a rădăcinii sale. Ea constă în scrierea etichetei ataşată nodului urmată, numai dacă are descendenţi, de scrierea parantezată a descendenţilor cuprinsă între paranteze.

 Pentru arborele de mai sus, scrierea parantezată este:

 **4(11(5,10),7,8(2,3,1(6)),9(12))**

1. Pentru un nod oarecare putem folosi următoarea reprezentare cu informaţiile alocate dinamic:
	* informaţia
	* adresa primului descendent (fiu) stâng
	* adresa primului frate drept

Această structură poate fi reprezentată astfel:

**typedef struct nod{**

 **int inf;**

 **nod \*fiu\_stang, \*frate\_drept;**

**}ARBORE;**

1. Pentru fiecare nod se poate stoca:
	* informaţia
	* numărul de descendenţi
	* adresa fiecărui descendent

Implementarea poate fi făcută astfel:

**typedef struct nod{**

 **int inf;**

 **int nr;** //numărul descendenţilor

 **nod \*fiu[NR\_MAXIM\_FII];** //tablou cu adresele descendenţilor

**}ARBORE;**

**3. Parcurgerea arborilor oarecare**

 Prin **parcurgere** se înţelege vizitarea în mod sistematic a nodurilor arborelui în scopul prelucrării informaţiei ataşate nodurilor sau a liniarizării nodurilor.

 Vom exemplifica metodele de parcurgere pe următorul arbore:

Un arbore oarecare poate fi parcurs astfel:

1. în **preordine**: se vizitează rădăcina şi apoi descendenţii direcţi ai fiecărui nod de la stânga la dreapta

Pentru arborele de mai sus, succesiunea preordine a nodurilor este:

 **4,11,5,10,7,8,2,3,1,6,9,12**

1. în **postordine**: se parcurg de la stânga la dreapta descendenţii direcţi ai fiecărui nod şi apoi se vizitează rădăcina

 Pentru arborele de mai sus, succesiunea postordine a nodurilor este:

 **5,10,11,7,2,3,6,1,8,12,9,4**

1. pe **nivele**: se parcurg în ordine nodurile de pe fiecare nivel de la stânga la dreapta, începând de la primul nivel la ultimul nivel:

 Pentru arborele de mai sus, succesiunea postordine a nodurilor este:

 **4,11,7,8,9,5,10,2,3,1,12,6**

După cum se observă definirea celor trei moduri de parcurgere este recursivă.

Pentru parcurgerea în **preordine** procedăm astfel: se vizitează nodul rădăcină şi se apelează recursiv funcţia de parcurgere pentru fiecare descendent, de la stânga la dreapta.

Pentru parcurgerea în **postordine** se apelează recursiv funcţia pentru parcurgerea tuturor descendenţilor şi apoi se va afişa informaţia nodului rădăcină.

Pentru parcurgerea pe **nivele** (orizontală) se utilizează o parcurgere de tip coadă pentru că este o parcurgere **BF** (în lăţime). Cât timp coada nu este vidă se extrage nodul curent din coadă şi se adaugă la capătul opus al cozii toţi descendenţii nodului.

Programul următor gestionează un arbore oarecare reprezentat prin metoda a treia descrisă în secţiunea anterioară: pentru fiecare nod se precizează informaţia ataşată, numărul de descendenţi şi tabloul cu adresele acestor descendenţi. Programul ilustrează modul de **creare** a unui astfel de arbore, cele trei metode de **parcurgere** descrise precum şi **ştergerea** din memorie a arborelui creat. Pentru ştergere se va proceda astfel: se şterg mai întâi toşi descendenţii unui nod, după care se şterge nodul tată. Ultimul nod şters va fi rădăcina arborelui.

**#include<iostream>**

**using namespace std;**

**#define NMAX 30** //numarul maxim de descendenti ai unui nod

**typedef struct nod{**

 **int inf;**

 **int n;** //numarul de descendenti

 **nod \*leg[NMAX];** //tabloul adreselor descendentilor

**}ARB;**

**ARB \*coada[100];** //coada pentru parcurgerea pe nivele

**int prim,ultim;** //pentru gestionarea cozii

**ARB\* Creare()** //creaza arborele oarecare si returneaza adresa radacinii

**{**

 **int info,nr,i; ARB \*p;**

 **cout<<"informatia nodului: "; cin>>info;**

 **p=new ARB; p->inf=info;**

 **cout<<"numarul descendentilor pentru "<<info<<": ";**

 **cin>>nr; p->n=nr;**

 //se descriu in ordine descendentii de la stanga la dreapta si in adancime

 **for(i=0;i<p->n;i++) p->leg[i]=Creare();**

//apelez recursiv functia pentru crearea descendentilor

 **return p;** //radacina arborelui

**}**

**void Preordine(ARB \*p)** //afiseaza nodurile in preordine

**{**

 **int i;**

 **if(p)**

 **{**

 **cout<<p->inf<<" ";** //afisez nodul tata

 **for(i=0;i<p->n;i++) Preordine(p->leg[i]);** //afisez descendentii

 **}**

**}**

**void Postordine(ARB \*p)** //afiseaza nodurile in postordine

**{**

 **int i;**

 **if(p)**

 **{**

 **for(i=0;i<p->n;i++)**

 **if(p->leg[i]) Postordine(p->leg[i]);** //afisez descendentii

 **cout<<p->inf<<" ";** //afisez nodul tata

 **}**

**}**

**void Adauga(ARB \*p)** //adauga un nod in coada

**{**

 **if(prim>ultim) cout<<"Coada este plina\n";**

 **else coada[ultim++]=p;**

**}**

**ARB\* Extrage\_nod()** //extrage un nod din coada

**{**

 **if(prim==ultim) return 0;**

 **else return coada[prim++];**

**}**

**void Traversare\_nivele(ARB \*rad)**

**{**

 **ARB \*p; int i;**

 **prim=ultim=0;**

 **Adauga(rad);** //in coada se introduce nodul radacina

 **do{**

 **p=Extrage\_nod();** //extrag un nod din coada

 **if(p)**

 **{**

 **cout<<p->inf<<" ";** //afisez informatia nodului

 **for(i=0;i<p->n;i++)**

 **Adauga(p->leg[i]);** //adaug in coada descendentii nodului

 **}**

 **}while(p);**

 **cout<<"\n";**

**}**

**void Sterge(ARB \*p)** //stergerea unui nod din arbore

**{**

 **int i;**

 **if(p)**

 **for(i=0;i<p->n;i++) Sterge(p->leg[i]);**

//mai intai sterg descendentii nodului

 **delete p;** //sterg nodul tata

**}**

**int main()**

**{**

 **ARB \*rad;** //radacina arborelui oarecare

 **cout<<"\n\t\tIntroduceti arborele:\n";**

 **rad=Creare(); //arborele a fost creat**

 **cout<<"\n\t\tTraversarea in preordine:\n";**

 **Preordine(rad);**

 **cout<<"\n\t\tTraversarea in postordine:\n";**

 **Postordine(rad);**

 **cout<<"\n\t\tTraversarea pe nivele:\n";**

 **Traversare\_nivele(rad);**

 **Sterge(rad);** //stergerea arborelui din heap

**return 0;**

**}**

**4. Definirea arborilor binari**

**Definiţie**: Un **arbore binar** este o mulţime finită de noduri care este fie vidă, fie reprezintă un arbore ordonat în care fiecare nod are cel mult doi descendenţi.

Un arbore binar conţine cel mult doi subarbori, pe care îi numim **subarbore stâng**, respectiv **subarbore drept**. Ei se pot obţine prin suprimarea rădăcinii şi a nodurilor incidente cu aceasta. Un nod fără descendenţi se numeşte **nod** **terminal** sau **frunză**.

Un arbore binar în care fiecare nod are 0 sau 2 descendenţi se numeşte **arbore binar complet**. Un astfel de arbore apare în figura de mai jos.

**Proprietăţi ale arborilor binari**

1. Un arbore binar complet care are **n** noduri terminale, toate situate pe acelaşi nivel, are în total **2n-1** noduri. În consecinţă, un arbore binar complet are un număr impar de noduri.
2. Numărul maxim de noduri de pe nivelul **i** al unui arbore este **2i**.
3. Numărul maxim de noduri dintr-un arbore binar cu înălţimea **h** este **2h+1-1**.
4. Un arbore binar cu **n** noduri are înălţimea mai mare sau egală cu **[log2n]**.

**5. Reprezentarea statică a arborilor binari**

Există mai multe posibilităţi de **reprezentare** a arborilor binari. În continuare sunt descrise trei metode de reprezentare statică a arborilor binari. Vom considera pentru exemplificare următorul arbore binar:

a) **Reprezentarea standard** se bazează pe următorul principiu: pentru fiecare nod în parte se precizează, dacă există, descendentul stâng şi descendentul drept. Dacă un nod este terminal, atunci acest lucru se precizează punând 0 în locul descendenţilor săi. Pentru aceast se utilizează fie doi vectori numiţi, de exemplu, **S**-pentru descendenţii din stânga şi **D**-pentru descendenţii din dreapta. Dacă pentru reprezentarea unui arbore binar cu **n** noduri se folosesc vectorii **S** şi **D**, atunci pentru fiecare nod **i∈{1,2,....n}** componenta **S[i]** conţine descendentul stâng al nodului **i**, iar componenta **D[i]** conţine descendentul drept al nodului **i**.

 De exemplu, pentru arborele binar de mai sus, cei doi vectori vor avea următoarea structură:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Nodul i** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **S[i]** | **2** | **4** | **0** | **0** | **6** | **0** | **8** | **0** | **0** |
| **D[i]** | **3** | **5** | **0** | **0** | **7** | **0** | **9** | **0** | **0** |

Se observă că nu este important să se precizeze rădăcina, deoarece ea nu este descendenta nici unui nod.

b) **Legături de tip tată**. Se folosesc doi vectori: **TATA** şi **DESC**. Pentru fiecare nod **i**, **TATA[i]** precizează care nod îi este ascendent (nodul părinte). **DESC[i]** poate lua două valori: **-1** dacă **i** este descendent stâng pentru **TATA[i]** şi **1** dacă este descendent drept pentru acesta. Pentru nodul rădăcină, care nu are un nod părinte asociat, valoarea corespunzătoare în cei doi vectori este 0. Pentru arborele binar de mai sus, configuraţia celor doi vectori este:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Nodul i** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **TATA[i]** | **0** | **1** | **1** | **2** | **2** | **5** | **5** | **7** | **7** |
| **DESC[i]** | **0** | **-1** | **1** | **-1** | **1** | **-1** | **1** | **-1** | **1** |

c) **Reprezentarea cu paranteze (parantezată**). Pentru a obţine o reprezentare a arborelui folosind paranteze, se procedează în felul următor:

1. se scrie nodul rădăcină
2. fiecare nod al arborelui va fi urmat de:
	* paranteză rotundă deschisă
	* descendent stâng
	* virgulă
	* descendent drept
	* paranteză rotundă închisă

Pentru arborele din figura anterioară, reprezentarea parantezată este: **1( 2 (4 , 5 ( 6 , 7 ( 8 , 9 ) ) ) , 3)**

Prin **parcurgerea arborilor binari** se înţelege, ca şi la grafurile obişnuite, examinarea în mod sistematic a nodurilor astfel încât fiecare nod să fie atins o singură dată. Această procedură se mai numeşte şi **vizitare** a nodurilor arborelui în scopul prelucrării informaţiei conţinută de fiecare dintre acestea. Deoarece arborii sunt o structură neliniară de date, rolul traversării este tocmai obţinerea unei aranjări liniare a nodurilor, pentru ca trecerea de la unul la altul să se realizeze cât mai simplu posibil. Există trei modalităţi de parcurgere a arborilor binari, toate utilizând modul standard de reprezentare (sau alocarea dinamică): în **preordine** (**R**ădăcină-**S**tânga-**D**reapta), în **inordine** (**S**tânga-**R**ădăcină-**D**reapta) şi în **postordine** (**S**tânga-**D**reapta-**R**ădăcină). Aceste trei sunt definite recursiv şi parcurgerea se face în trei etape:

**a) Traversarea în preordine (RSD)**

* se vizitează rădăcina
* traversează subarborele stâng
* traversează subarborele drept

**b) Traversarea în inordine (SRD)**

* se traversează subarborele stâng
* se vizitează rădăcina
* se traversează subarborele drept

**c) Traversarea în postordine (SDR)**

* se traversează subarborele stâng
* se traversează subarborele drept
* se vizitează rădăcina

Pentru arborele binar din figura de mai sus, avem următoarele rezultate ale parcurgerilor:

* metoda **RSD**: **1, 2, 4, 5, 6, 7, 8, 9, 3**
* metoda **SRD**: **4, 2, 6, 5, 8, 7, 9, 1, 3**
* metoda **SDR**: **4, 6, 8, 9, 7, 5, 2, 3, 1**

Programul care implementează cele trei metode de parcurgere (bazate pe o strategie DEI) este prezentat în continuare:

**#include<iostream>**

**using namespace std;**

**#define N 30**

**int S[N],D[N],rad,n;**

**void SRD(int k)** //parcurgere inordine

**{**

 **if(S[k]) SRD(S[k]);**

 **cout<<k<<" ";**

 **if(D[k]) SRD(D[k]);**

**}**

**void RSD(int k)** //parcurgere preordine

**{**

 **cout<<k<<" ";**

 **if(S[k]) RSD(S[k]);**

 **if(D[k]) RSD(D[k]);**

**}**

**void SDR(int k)** //parcurgere postordine

**{**

 **if(S[k]) SDR(S[k]);**

 **if(D[k]) SDR(D[k]);**

 **cout<<k<<" ";**

**}**

**int main()**

**{**

 **int i;**

 **cout<<"numar varfuri, n=";**

 **do{**

 **cin>>n;**

 **}while(n<1||n>N);**

 **cout<<"radacina arborelui: ";**

 **do{**

 **cin>>rad;**

 **}while(rad<1||rad>n);**

 //memorarea arborelui prin vectorii S,D

 **for(i=1;i<=n;i++)**

 **{**

 **cout<<"descendentii stanga,dreapta ("<<i<<"): ";**

 **cin>>S[i]>>D[i];**

 **}**

 **cout<<"\n\tParcurgerea in preordine:\n";**

 **RSD(rad);**

 **cout<<"\n\n\tParcurgerea in inordine:\n";**

 **SRD(rad);**

 **cout<<"\n\n\tParcurgerea in postordine:\n";**

 **SDR(rad);**

 **cout<<"\n";**

**return 0;**

**}**

**6. Alocarea dinamică a arborilor binari**

În acest mod de alocare, fiecare nod este o structură cu trei câmpuri: informaţia ataşată nodului, adresa fiului stâng şi adresa fiului drept. Absenţa unui fiu este marcată cu pointerul nul. Se observă că acest mod de reprezentare este similar cu reprezentarea standard de la alocarea statică a arborilor binari. Pentru descrierea unui nod putem folosi declaraţiile:

 **typedef struct nod{**

 **int inf;** //sau orice alt tip pentru informaţie

 **nod \*st,\*dr;** //adresele fiilor

 **}ARB;**

**Crearea unui arbore binar** alocat dinamic se realizează cel mai uşor aplicând metoda DEI astfel:

* se generează un nod, adică se alocă spaţiu în heap şi se încarcă informaţia
* pentru fiecare nod se construieşte subarborele său stâng, apoi subarborele său drept şi se completează adresele descendenţilor nodului cu adresele acestor subarbori
* un descendent vid trebuie marcat printr-o proprietate stabilită asupra informaţiei (spre exemplu apariţia valorii 0 drept conţinut informaţional al nodului curent)

Spre exemplu, pentru arborele binar următor

şirul datelor furnizat la intrare va arăta astfel: **1 2 4 0 0 5 7 0 0 0 3 0 6 8 0 0 9 0 0** ceea ce corespunde unei liste în preordine a nodurilor.

Funcţia care crează un arbore binar furnizat ca mai sus este:

**typedef struct nod{**

 **int inf;**

 **nod \*st,\*dr;**

**}ARB;**

**void Creare(ARB\* &r)**

**{**

 **int x;**

 **cin>>x;**

 **if(!x)** //urmeaza subarbore vid

 **r=0;**

 **else**

 **{**

 **r=new ARB; r->inf=x;**

//creare nod curent si memorare informatie

 **Creare(r->st); Creare(r->dr);** //crearea subarborilor

 **}**

**}**

Funcţia **Creare()** se poate proiecta astefel încât să întoarcă adresa rădăcinii arborelui. În acest caz, funcţia va avea antetul: **ARB\* Creare(ARB\* &r)**.

**Parcurgerea arborilor binari** poate fi făcută:

* în adâncime (pe subarbori) în trei moduri: ***preordine*** (**RSD**), ***inordine*** (**SRD**), ***postordine***(**SDR**)
* în lăţime (pe niveluri, ca la arborii oarecare)

Programul următor realizează crearea unui arbore binar şi afişează listele nodurilor obţinute pein cele trei metode de parcurgere în adâncime.

**#include<iostream>**

**using namespace std;**

**typedef struct nod{**

 **int inf;**

 **nod \*st,\*dr;**

**}ARB;**

**void Creare(ARB\* &r)**

**{**

 **int x;**

 **cin>>x;**

 **if(!x)** //urmeaza subarbore vid

 **r=0;**

 **else**

 **{**

 **r=new ARB; r->inf=x;**

//creare nod curent si memorareinformatie

 **Creare(r->st); Creare(r->dr);** //crearea subarborilor

 **}**

**}**

**void SRD(ARB \*r)** //parcurgere inordine

**{**

 **if(r)**

 **{**

 **SRD(r->st); cout<<r->inf<<" "; SRD(r->dr);**

 **}**

**}**

**void RSD(ARB \*r)** //parcurgere preordine

**{**

 **if(r)**

 **{**

 **cout<<r->inf<<" "; RSD(r->st); RSD(r->dr);**

 **}**

**}**

**void SDR(ARB \*r)** //parcurgere postordine

**{**

 **if(r)**

 **{**

 **SDR(r->st); SDR(r->dr); cout<<r->inf<<" ";**

 **}**

**}**

**void Sterge(ARB \*r)**

**{**

 **if(r)**

 **{**

 **Sterge(r->st); Sterge(r->dr); delete r;**

 **}**

**}**

**int main()**

**{**

 **ARB \*rad;** //adresa radacinii arborelui

 **cout<<"Introduceti nodurile : ";**

 **Creare(rad);** //crearea arborelui binar

 **cout<<"\nNodurile in inordine: "; SRD(rad);**

 **cout<<"\nNodurile in preordine: "; RSD(rad);**

 **cout<<"\nNodurile in postordine: "; SDR(rad);**

 **cout<<"\n";**

 **Sterge(rad);** //stergerea arborelui din heap

**return 0;**

**}**

**7. Forma poloneză a expresiilor aritmetice**

Se mai numeşte ***notaţia fără paranteze a expresiilor aritmetice*** şi de studiul acestei probleme s-a ocupat matematicianul polonez J. Lukasiewicz. Deoarece într-o expresie aritmetică, un operator se aplică unui operand sau unei perechi de operanzi, s-a ajuns la ideea de a asocia unei expresii aritmetice un arbore binar. Presupunem că se notează cu **E1** respectiv cu **E2** două expresii aritmetice cărora li se aplică operatorul notat “**op**”. Operatorii pe care îi vom accepta sunt doar operatorii binari, şi anume: adunarea **(+),** scăderea **(-),** înmulţirea **(\*),** împărţirea **(/)** şi ridicarea la putere **(^).** Pentru a asocia unei expresii aritmetice fără paranteze un arbore binar se folosesc următoarele reguli:

* unei expresii aritmetice formată dintr-un singur operand i se asociază un arbore binar format doar din nodul rădăcină în care se pune operandul respectiv
* dacă expresia aritmetică **E** este de forma **E1 op E2**, atunci arborele binar complt asociat are ca rădăcină operatorul **op**, ca subarbore stâng are arborele binar asociat expresiei aritmetice **E1**, iar ca subarbore drept are arborele binar asociat expresiei aritmetice **E2**

**Exemplu**: Fie expresiile cele mai simple care se pot forma cu operatorii binari cunoscuţi: **a+b**, **a-b**, **a\*b**, **a/b** şi **ab**. Considerând op unul dintre operatorii binari **+,-,\*,/,∧**, arborele binar asociat este cel din figura următoare:

**Exemplu**: Fie expresia **E=a\*b+c/d-e**. Ne propunem să determinăm arborele binar asociat. Se observă că, deoarece nu este stabilit modul de asociere a termenilor, se pot obţine doi arbori binari, după cum se consideră **E1=a\*b+c/d** şi **E2=e**, sau **E1=a\*b** şi **E2=c/d-e**. Se obţin corespunzător arborii din figura de mai jos.

Aceşti arbori binari se pot parcurge folosind oricare din cele trei metode. Se obţine astfel:

**RSD: -+\*ab/cde respectiv**

**RSD: +\*ab-/cde**

**SRD: a\*b+c/d-e respectiv**

**SRD: a\*b-c/d-e**

**SDR: ab\*cd/+e- respectiv**

**SDR: ab\*cd/e-+**

Şirul obţinut prin parcurgerea în preordine a arborelui binar ataşat expresiei aritmetice se numeşte **forma poloneză prefixată** asociată expresiei, iar şirul obţinut prin prin parcurgerea în postordine se numeşte **forma poloneză postfixată** asociată expresiei aritmetice.

**Observaţii**: Se poate construi arborele binar asociat şi pentru o expresie aritmetică care conţine paranteze rotunde. În forma poloneză nu apar parantezele, acestea nemaifiind necesare pentru a marca priorităţile de calcul.

 Programul următor citeşte o expresie aritmetică care conţine paranteze rotunde, operanzi de o literă şi operatori binari **+,-,\*,/** şi construieşte forma poloneză prefixată. Pentru a construi în memorie arborele binar ataşat pornind de la expresia aritmetică (citită ca şir de caractere) se acordă priorităţi operatorilor şi operanzilor astfel:

* prioritatea iniţială a operatorilor +,- este 1
* prioritatea iniţială a operatorilor \*,/ este 10
* la prioritatea unui operand se adaugă zece unităţi pentru fiecare pereche de paranteze între care se găseşte
* prioritatea unui operand este 1000

Construirea priorităţilor se realizează astfel:

* se citeşte expresia aritmetică în şirul de caractere **e**
* se utilizează o variabilă **j** care indică ce valoare se adaugă la prioritatea iniţială a unui operator (la întâlnirea unei paranteze deschise **j** creşte cu 10, iar la întâlnirea unei paranteze închise scade cu 10)
* se parcurge expresia caracter cu caracter şi se memorează în vectorul **p** priorităţile operatorilor şi operanzilor
* în vectorul **efp** se construieşte expresia aritmetică fără paranteze, iar în **pfp** se obţine vectorul priorităţilor din care, spre deosebire de **epf**, lipsesc componentele corespunzătoare parantezelor
* utilizând vectorii **efp** şi **pfp** funcţia **Creare()** construieşte în heap arborele binar ataşat expresiei aritmetice.

Conform tehnicii ***Divide Et Impera***, funcţia **Creare()** lucrează astfel:

* de la limita superioară (**ls**) către cea inferioară (**li**), limite corespunzătoare şirurilor din **efp**, se caută operandul sau operatorul cu prioritate minimă, reţinându-se şi poziţia acestuia
* în situaţia în care limita superioară este diferită de cea inferioară, pentru completarea adresei subarborelui stâng, respectiv drept, se reapelează funcţia **Creare()**

**#include<iostream>**

**#include<string.h>**

**using namespace std;**

**#define MAX 100**

**typedef struct nod{**

 **char inf;**

 **nod \*st,\*dr;**

**}ARB;**

**char e[MAX],efp[MAX];**

**int p[MAX],pfp[MAX];**

**ARB \*rad;**

**void Creare(ARB\* &c,int li,int ls,char epf[],int pfp[])**

**{**

 **int i,j,min;**

 **min=pfp[ls]; i=ls;**

 **for(j=ls;j>=li;j--)**

 **if(pfp[j]<min) { min=pfp[j]; i=j; }**

 **c=new ARB; c->inf=efp[i];**

 **if(li==ls)**

 **c->st=c->dr=0;**

 **else**

 **{**

 **Creare(c->st,li,i-1,efp,pfp);**

 **Creare(c->dr,i+1,ls,efp,pfp);**

 **}**

**}**

**void Parcurgere(ARB \*c)**

**{**

 **if(c)**

 **{**

 **cout<<c->inf;Parcurgere(c->st); Parcurgere(c->dr);**

 **}**

**}**

**int main()**

**{**

 **int i,j=0;**

 **cout<<"introduceti expresia: "; gets(e);**

 **for(i=0;e[i];i++)**

 **switch(e[i])**

 **{**

 **case ')': j-=10; break;**

 **case '(': j+=10; break;**

 **case '+':**

 **case '-': p[i]=j+1; break;**

 **case '\*':**

 **case '/': p[i]=j+10; break;**

 **default: p[i]=1000;**

 **}**

 **j=-1;**

 **for(i=0;e[i];i++)**

 **if(e[i]!=')'&&e[i]!='(')**

 **{**

 **j++; efp[j]=e[i]; pfp[j]=p[i];**

 **}**

 **Creare(rad,0,j,efp,pfp);**

 **cout<<"\nForma poloneza prefixata este: ";**

 **Parcurgere(rad); cout<<"\n";**

**return 0;**

**}**

Modul în care se foloseşte forma poloneză prefixată pentru evaluarea expresiei aritmetice asociate este următorul: se parcurge forma poloneză prefixată de la dreapta la stânga utilizându-se o stivă pentru memorarea operanzilor. Dacă simbolul curent este un operand, el se introduce în vârful stivei. Dacă simbolul citit este un operator, se aplică acest operator primilor doi operanzi din vârful stivei obţinându-se un rezultat. Cei doi operanzi sunt eliminaţi din stivă şi în locul lor se trece rezultatul obţinut. Se trece apoi la următorul simbol care conţine forma poloneză prefixată asociată expresiei aritmetice.

Programul următor ilustrează această modalitate de evaluare a expresiilor aritmetice. Operanzii sunt identificaţi printr-o singură literă, iar operatorii sunt binari din mulţimea **+,-,\*,/**.

**#include<iostream>**

**#include<string.h>**

**using namespace std;**

**char e[100];**

**float s[100],x,y;** //s este stiva

**int n,i,p;**

**int main()**

**{**

 **cout<<"\tIntroduceti expresia aritmetica in forma prefixata\n";**

 **gets(e); n=strlen(e);**

 **for(i=n-1;i>=0;i--)**

 **if(e[i]=='+'||e[i]=='-'||e[i]=='\*'||e[i]=='/')**

 **{**

 **x=s[p--]; y=s[p];**

 **switch(e[i])**

 **{**

 **case '+': s[p]=x+y; break;**

 **case '-': s[p]=x-y; break;**

 **case '\*': s[p]=x\*y; break;**

 **case '/': s[p]=x/y; break;**

 **}**

 **}**

 **else**

 **s[++p]=e[i]-'0';**

 **cout<<"\nvaloarea expresiei este "<<s[1]<<endl;**

**return 0;**

**}**