Uređeno stablo (Tree)

Uređeno stablo T je neprazan konačan skup podataka istog tipa (čvorova) takav da

(i) postoji jedan istaknuti čvor (korijen stabla)

(ii) skup ostalih čvorova (ako je neprazan) particioniran je na skupove T_1, T_2, ... , T_k od kojih je svaki opet uređeno stablo

Uvodimo pojmove podstabla, roditelja, djece i braće čvora, puta i duljine puta, visine stabla, predaka i potomaka čvora, nivoa, lista i unutrašnjeg čvora.

Primjer.

listovi:
korijen:
roditelj od C:
djeca od C:
preci od E:
potomci od E:
idući brat od D:
idući brat od E:
čvorovi na nivou 2:
visina stabla:

Osnovni obilasci stabla

PREORDER: korijen T_1 T_2 ... T_k

INORDER: T_1 korijen T_2 ... T_k

POSTORDER: T_1 T_2 ... T_k korijen

a.t.p. Tree

`node`	. . .	tip podatka kojim je jednoznačno određen čvor, ime čvora (usp. `position` kod a.t.p. List); u skupu `node` je poseban element `LAMBDA` koji služi kao ime nepostojećeg čvora
`labeltype`	. . .	tip oznake čvora, tj. tip korisne informacije koju čuvamo u čvoru
`Tree`	. . .	uređeno stablo čiji čvorovi tipa `node` sadrže podatke tipa `labeltype`
`TrMakeRoot(l,&T)`	. . .	pretvara stablo `T` u stablo koje se sastoji samo od korijena s oznakom `l`, vraća ime korijena
`TrInsertChild(l,i,&T)`	. . .	u stablo `T` ubacuje novi čvor s oznakom `l` tako da on bude prvo dijete čvora `i`, vraća ime novog čvora
`TrInsertSibling(l,i,&T)`	. . .	u stablo `T` ubacuje novi čvor s oznakom `l` tako da on bude idući brat čvora `i`, vraća ime novog čvora
`TrDelete(i,&T)`	. . .	briše list s imenom `i` iz stabla `T`
`TrRoot(T)`	. . .	vraća ime korijena stabla `T`
`TrFirstChild(i,T)`	. . .	vraća ime prvog djeteta čvora `i` u stablu `T` (ako takvo ne postoji, vraća `LAMBDA`)
`TrNextSibling(i,T)`	. . .	vraća ime idućeg brata čvora `i` u stablu `T` (ako takav ne postoji, vraća `LAMBDA`)
`TrParent(i,T)`	. . .	vraća ime roditelja čvora `i` u stablu `T` (ako je `i` korijen, vraća `LAMBDA`)
`TrLabel(i,T)`	. . .	vraća oznaku čvora `i` u stablu `T`
`TrChangeLabel(l,i,&T)`	. . .	mijenja oznaku čvora `i` u stablu `T` u oznaku `l`

Zadatak 1.

Napišite funkciju koja vraća visinu stabla neovisno o implementaciji a.t.p. Tree.

Rješenje. Rekurzijom. Visina stabla T jednaka je 0 ako stablo ima samo jedan čvor (korijen), a inače je jednaka $1 + \max\{\text{visina stabla } T_i \mid i = 1, ..., k\}$.

```
int Height(node i, Tree T) {
    int max = 0;
    node child;
    for (child = TrFirstChild(i, T); child != LAMBDA; child = TrNextSibling(child, T)) {
        int visina = Height(child, T);
        if (visina + 1 > max) max = visina + 1;
    }
    return max;
}
```

Visinu cijelog stabla dobijemo pozivom funkcije `Height(Root(T),T)`.

Zadatak 2.

Napišite nerekurzivnu verziju PREORDER obilaska stabla, neovisno o implementaciji a.t.p. Tree i bez upotrebe funkcije `Parent`.

Rješenje.

```
void PreorderStack(Tree T) {
    node tren;
    Stack S;    /* elementtype je node, u stogu pamtimo put
                           od korijena do trenutacnog cvora */
    StMakeNull(&S);
    tren = TrRoot(T);
    while (1) {
        if (tren != LAMBDA) {
            printf("%d ", TrLabel(tren, T));
            StPush(tren, &S);
            tren = TrFirstChild(tren, T);
        }
        else {
            if (StEmpty(S))
                break;
            tren = TrNextSibling(StTop(S), T);
            StPop(&S);
        }
    }
}
```

Napomena. Na predavanjima su obrađene implementacije stabla pomoću kursora koje se zasnivaju na vezama čvor $\rightarrow$ roditelj ili čvor $\rightarrow$ (prvo dijete, idući brat). U obje implementacije pojedine funkcije iz a.t.p. Tree imaju preveliku složenost (linearnu). Kombinacijom tih implementacija sve funkcije iz a.t.p. Tree složenosti su $O(1)$:

```
typedef int node;
const node LAMBDA = -1;

struct celltype {
    labeltype label;
    node first_child, next_sibling, parent;
} Space[MAX_CELLS];

typedef int Tree;  /* kursor na korijen */
```

Možemo i više stabala prikazati u polju `Space`; svi neiskorišteni indeksi povezani su u vezanu listu (preko naprimjer `next_sibling`) na čiji početak imamo globalni kursor `avail`.

Primjer. Prikazat ćemo stablo sa slike u polju `Space` s `MAX_CELLS` $= 16$.

Zadatak 3.

U kontekstu prethodno opisane implementacije stabla napišite funkciju koja vraća PREORDER obilazak stabla, nerekurzivno i bez stoga.

Rješenje. Razlika je u tome što sada ne moramo pamtiti put – roditelj je uvijek dostupan.

```
void PreorderParent(int T) {
    int tren = T, roditelj = -1;
    while(1) {
        if (tren != -1) {
            printf("%d ", Space[tren].label);
            roditelj = tren;
            tren = Space[tren].first_child;
        }
        else {
            if (roditelj == -1) break;
            tren = Space[roditelj].next_sibling;
            roditelj = Space[roditelj].parent;
        }
    }
}
```