Stog (Stack)

Stog je posebna vrsta liste: od svih operacija dozvoljeno je ubacivanje, brisanje i gledanje sadržaja elementa samo na jednom kraju liste koji zovemo *vrh stoga*. Stog zovemo i LIFO – last in first out.

a_n ← vrh stoga
$\vdots$
a_2
a_1
a_0

a.t.p. Stack

`elementtype`	. . .	bilo koji tip
`Stack`	. . .	konačan niz podataka tipa `elementtype`
`StMakeNull(&S)`	. . .	pretvara `S` u prazan stog
`StEmpty(S)`	. . .	vraća `1` ako je `S` prazan, `0` ako nije
`StPush(x,&S)`	. . .	ubacuje element `x` na vrh stoga `S`
`StPop(&S)`	. . .	uklanja element s vrha stoga; nije definirano ako je `S` prazan
`StTop(S)`	. . .	vraća element koji je na vrhu stoga (ne uklanja ga); nije definirano ako je `S` prazan

Primjer. $\underset{S}{\ } \rightarrow$ `Push(1,&S)` $\rightarrow \underset{S}{1} \rightarrow$ `Push(2,&S)` $\rightarrow \underset{S}{\overset{2}{1}} \rightarrow$ `Pop(&S)` $\rightarrow \underset{S}{1} \rightarrow$ `Push(3,&S)` $\rightarrow \underset{S}{\overset{3}{1}}$

Uvijek trebamo paziti da ne zovemo `Pop` nad praznim stogom.

Upotreba. Svaki poziv funkcije u programskim jezicima stavlja povratnu adresu i parametre na stog. Stog nam omogućava simulaciju rekurzivnog programa nerekurzivnim.

Primjer. Izvrednjavanje POSTFIX aritmetičkih izraza.
INFIX operand1 operator operand2 ≡ POSTFIX operand1 operand2 operator
Primjer evaluacije na stogu: `2 3 + 5 6 * -` → `5 5 6 * -` → `5 30 -` → `-25`

Zadatak 5.

Opišite evaluaciju POSTFIX izraza na stogu. Prikažite sve korake na primjeru izračunavanja izraza `(A/(B^C))*D+E` (prebacite ga prvo u POSTFIX oblik).

Rješenje. POSTFIX oblik izraza je: `A B C ^ / D * E +` .
Algoritam za evaluaciju POSTFIX izraza:

```
za svaki znak ch postfix izraza redom {
    ako je ch operand,
        stavi ch na stog;
    inace {
        uzmi broj s vrha stoga i spremi ga u b2 (ako je stog prazan, javi gresku);
        uzmi broj s vrha stoga i spremi ga u b1 (ako je stog prazan, javi gresku);
        izvrsi operaciju ch nad b1 i b2;
        rezultat stavi na stog;
    }
}
uzmi broj s vrha stoga, to je rezultat;
```

Slijed operacija i vrijednosti na stogu u svakom koraku:

Napomena. Kako automatizirati prevođenje INFIX oblika u POSTFIX oblik? Dijkstrinim algoritmom. Uočimo probleme: prioritet operatora (npr. INFIX `1+2*3` nije POSTFIX `12+3*` već `123*+`) i zagrade (no one se lako rješavaju rekurzivno). Prioriteti operatora su: najveći prioritet ima $\uparrow$, zatim idu $*$ / i najmanji prioritet imaju $+$ $-$.

DIJKSTRIN ALGORITAM

Ulaz: izraz u INFIX obliku. Algoritam:

```
za svaki znak ch infix izraza {
    ako je ch broj, ispisi ga;
    ako je ch operator {
        sve dok stog nije prazan i operator na vrhu ima veci ili jednak prioritet od ch
            ispisi i ukloni operator na vrhu stoga;
        stavi ch na stog;
    }
    ako je ch == '(', stavi ch na stog;
    ako je ch == ')' {
        ispisuj i uklanjaj sve operatore s vrha stoga sve dok ne naidjes na '(';
        '(' samo ukloni;
    }
}
ispisuj i uklanjaj sve operatore s vrha stoga dok se on ne isprazni;
```

(Elegantnije: ako na početku stavimo izraz u zagrade, zadnja linija neće biti potrebna.) Dakle, na stog uvijek stavljamo točno one operatore koji imaju veći prioritet od operatora na stogu.

Zadatak 6.

Dijkstrinim algoritmom prebacite `A+B*C+(D+E)*F` u POSTFIX.

Rješenje.

POSTFIX: `ABC*+DE+F*+`

Zadatak 7.

Problem Hanojskih tornjeva glasi ovako: Dana su tri štapa A, B i C. Na štap A nanizano je n diskova različitih promjera.

Potrebno je prebaciti sve diskove sa štapa A na štap C koristeći se štapom B kao pomoćnim. Odjednom je moguće prebaciti samo jedan disk i ne smije se stavljati veći disk na manji. Napišite rekurzivni program koji rješava ovaj problem.

Rješenje.

```
void Hanoi(int n, char pocetni, char pomocni, char zavrsni) {
    if (n == 1) {
        move(pocetni, zavrsni);
        return;
    }
    Hanoi(n-1, pocetni, zavrsni, pomocni);
    move(pocetni, zavrsni);
    Hanoi(n-1, pomocni, pocetni, zavrsni);
    return;
}
```

Zadatak 8.

Napišite nerekurzivnu verziju funkcije `Hanoi`.

Rješenje. Rekurzivne pozive simuliramo pomoću stoga (neovisno o implementaciji). Potrebno je na stog spremiti trenutni opis situacije (n, koji štap je početni, koji pomoćni, ...), potrebno je i znati u kojoj smo fazi (time simuliramo adresu instrukcije na koju se vraćamo nakon funkcijskog poziva).

```
typedef struct {
    int n, faza;
    char poc, pom, zav;
} elementtype;

void HanoiStack(int n, char poc, char pom, char zav) {
    elementtype x;
    Stack S;
    StMakeNull(&S);
    f1: if (n == 1) {
            move(poc, zav);
            goto f5;
        }
    f2: //spremamo trenutnu situaciju na stog
        x.n = n; x.poc = poc; x.pom = pom; x.zav = zav; x.faza = 3;
        StPush(x, &S);
        //pripremimo parametre
        n = x.n - 1; poc = x.poc; pom = x.zav; zav = x.pom;
        //simuliramo poziv funkcije s novim parametrima
        goto f1;
    f3: move(poc, zav);
    f4: //spremamo trenutnu situaciju na stog
        x.n = n; x.poc = poc; x.pom = pom; x.zav = zav; x.faza = 5;
        StPush(x, &S);
        //pripremimo parametre
        n = x.n - 1; poc = x.pom; pom = x.poc; zav = x.zav;
        //simuliramo poziv funkcije s novim parametrima
        goto f1;
    f5: if StEmpty(S)
            return;
        else {
            //gotovi smo s jednim pozivom, treba se vratiti na nivo iznad
            //uzimamo stanje nivoa sa stoga
            x = StTop(S); StPop(&S);
            n = x.n; poc = x.poc; pom = x.pom; zav = x.zav;
            //vracamo se na sljedecu naredbu u tom nivou
            switch (x.faza) {
                case 3: goto f3; break;
                case 5: goto f5; break;
            }
        }
}
```

Zadatak za DZ

Razradite implementaciju stoga pomoću pointera. Napišite sve potrebne definicije tipova i potprograme za operacije.