PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **14** (1986/1987)
Številka 6
Strani 305–309, 333

Matija Lokar:

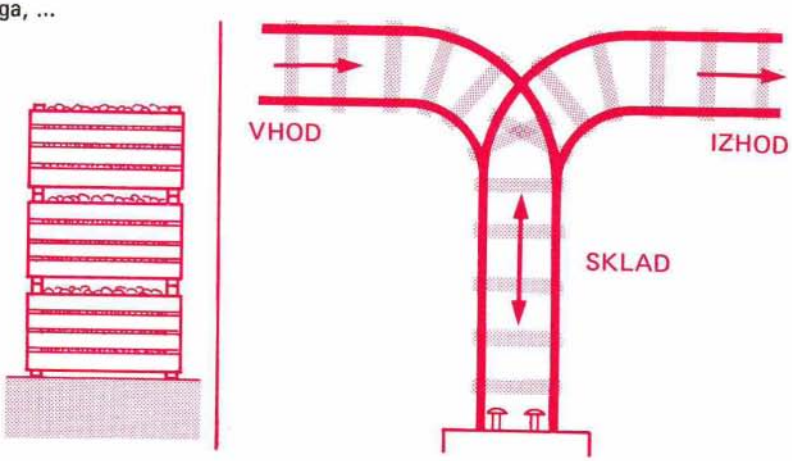
PODATKOVNA STRUKTURA SKLAD

Ključne besede: matematika, računalništvo, podatkovne strukture, sklad.

Elektronska verzija: http://www.presek.si/14/859-Lokar.pdf

PODATKOVNA STRUKTURA SKLAD

S pojmom sklad se v vsakdanjem življenju dokaj pogosto srečamo. Tako se v jeseni v kleteh pojavijo zabojčki s krompirjem in jabolki, zloženi drug na drugem v sklad, železničarji imenujejo sklad razporeditev tirov, ki jim omogoča prestavljanje vagonov, v kuhinjski omarici so krožniki zloženi eden vrh drugega, ...



Tudi v računalništvu se pogosto srečamo s pojmom sklada. Pravimo, da je sklad ena osnovnih **podatkovnih struktur.** In kaj sploh je podatkovna struktura? Najkrajše jo lahko označimo kot skupino podatkov, nad katerimi lahko izvajamo določeno operacije. Definicija sicer ni povsem natančna, tudi pomanjkljiva je, a bo za našo rabo povsem zadostna.

Če hočemo torej predstaviti sklad, moramo ugotoviti, katere so operacije, ki so za sklad najbolj značilne. Odpravimo se zato v skladišče, kjer je gotovo veliko skladov zabojev. Postavimo se tja v kot, da ne bomo v napoto, in opazujmo. Pripeljal je tovornjak, naložen s štedilniki. Viličarji ga razkladajo. Prvi zaboj gre na tla, drugega položijo nanj, na tega spet tretjega, ... Mimo pride skladiščnik in nekaj razburjeno vpije. Nekaj je narobe. Aha! Štedilnik, ki so ga naložili kot drugega, ne spada sem, ker je druge vrste. Kaj pa sedaj? Kar ven se ga ne da potegniti, saj so nad njim naloženi še trije štedilniki. Zato viličar odstrani zgornji štedilnik, ki je bil naložen zadnji, nato naslednjega, še enega in šele sedaj pride na vrsto napačni zaboj. En viličar ga odpelje, drugi pa na-

loži preostale tri zaboje v sklad. Vidimo, da zaboje jemljejo iz sklada ravno v obratnem vrstnem redu, kot so jih nanj nalagali. In to je tudi glavna značilnost sklada: element, ki prej pride v sklad, ga zapusti kasneje. Zato v tuji literaturi sklad imenujejo tudi LIFO seznam. LIFO je kratica za **L**ast **I**n **F**irst **O**ut. Sklad je torej seznam, kjer element, ki gre zadnji "noter", pride prvi "ven". Pa še eno značilnost sklada smo lahko opazili v skladišču. Vse, kar se s skladom dogaja, se dogaja na enem koncu. Na istem koncu nalagamo elemente, z istega konca elemente jemljemo. Zato ta konec zasluži posebno ime. Imenujemo ga **vrh** sklada.

Katere bodo torej operacije, ki jih bomo izvajali nad skladom? Najprej si moramo prostor za sklad pripraviti. Ustezno operacijo bomo imenovali kar PRIPRAVI. V sklad vstavljamo in iz njega jemljemo elemente. To bosta izvajali operaciji VSTAVI in ODSTRANI. Seveda lahko iz sklada jemljemo elemente le toliko časa, dokler ta ni prazen. S pogojem PRAZEN bomo kontrolirali, če je v skladu še kaj elementov. Pogosto nas zanima tudi to, kateri element je na vrhu sklada. To operacijo bomo imenovali VRH.

Operacije smo tako določili. Še vedno pa nismo nič rekli o tem, zakaj sklad uporabljamo v računalniku in kako ga v računalniku sploh predstavimo. Odložimo to še za trenutek in poglejmo, kako bi formalno opisali podatkovno strukturo sklad.

V prvem delu predstavitve strukture, označenem s **declare,** naštejemo operacije, ki jih lahko nad to podatkovno strukturo izvajamo, v drugem delu, označenem z **where,** pa bomo našteli pravila, ki jih morajo te operacije upoštevati. Kakorkoli potem predstavimo sklad v računalniku in na kakršenkoli način te operacije sestavimo kot procedure, mora ta predstavitev zadoščati naštetim pravilom. Omenimo le operacijo **"Prazen",** katere rezultat je logična vrednost, ki jo bomo označili z Boolean. Oznako za ta tip si bomo sposodili iz programskega jezika pascal. Izraz tipa Boolean lahko zavzame dve vrednosti, pravilno (true) ali napačno (false).

Sklad je tako definiran in ga lahko uporabimo. Vrste uporabe sklada so v računalništvu številne. Tako brez sklada ne moremo speljati rekurzije, uporabljamo ga pri izračunu izrazov, služi nam za označevanje vrstnega reda obdelovanja podatkov, kjer moramo določene korake opraviti šele kasneje, ko so izpolnjeni drugi pogoji. Poglejmo si primer. Denimo, da izvajamo nalogo A. Med izvajanjem pa pridemo do točke, ko moramo najprej opraviti nalogo B, da bi z nalogo A lahko nadaljevali. Zato se lotimo naloge B. Še prej pa si moramo zapomniti, kje moramo nadaljevati, ko bomo končali z nalogo B. Zato v sklad shranimo potrebne podatke iz naloge A. Pri opravljanju naloge B pa moramo spet najprej izvesti nalogo C. Spet shranimo ustrezno informacijo v sklad

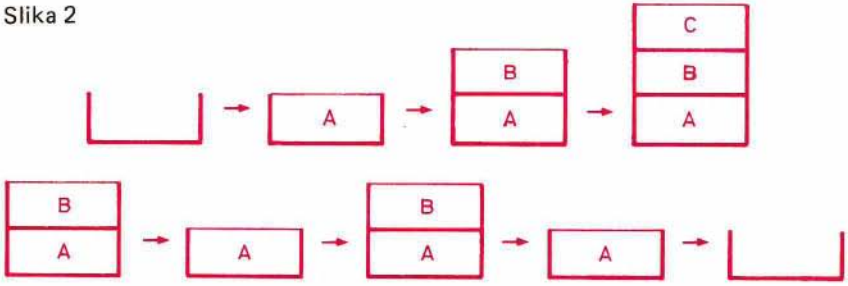
```
structure sklad
(* struktura "sklad" nad poljubno zalogo vrednosti tipa "podatek"*)
begin
    declare
        (*pripravi prazen sklad *)
        pripravi: Ø ⇒ sklad;
        (*vstavi podatek v sklad in vrne nov sklad *)
        vstavi: (podatek, sklad) ⇒ sklad,
        (* vrne podatek, ki je na vrhu sklada *)
        vrh : sklad ⇒podatek;
        (* izbriše vrhnji element sklada in vrne novi sklad *)
        odstrani: sklad ⇒ sklad;
        (* ali je sklad prazen *)
        prazen: sklad ⇒ Boolean;
    where
        prazen (pripravi) := true;
        prazen (vstavi (p,s)) := false;
        odstrani (pripravi) := napaka;
        odstrani (vstavi (p,s)) := s;
        vrh (pripravi) := napaka;
        vrh (vstavi (p,s)) := p;
end.
```

in pričnemo izvajati nalogo C. Po nekaj korakih ta spet zahteva, da najprej opravimo nalogo D. V sklad shranimo podatke o C in se lotimo naloge D. To uspešno končamo. Kaj pa sedaj? Edina naloga, ki jo lahko izvajamo, je naloga C, saj A zahteva, da je končana naloga B, le—ta pa, da je končana naloga C. Nalogo C vzamemo z vrha sklada in izvedemo do konca. Nadaljujemo z izvajanjem naslednje naloge na vrhu sklada. To je naloga B. Vendar po nekaj korakih le—ta spet zahteva, da dokončamo še nalogo E. Zato B ponovno uvrstimo v sklad in izvedemo nalogo E. Ko to končamo, se ponovno lotimo vrhnje naloge v skladu, B, in ko je opravljena, iz sklada vzamemo še zadnjo nalogo A in jo dokončamo. Spreminjanje sklada je prikazano na sliki 2.

Vidimo, da v vsakem trenutku med obdelavo sklad avtomatično vzdržuje vrstni red, potreben za uspešno izvajanje naloge. Na ta način računalnik tudi nadzoruje izvajanje programa, če je A glavni program, B, C, D in E pa ustrezni podprogrami.

Ostane nam le še, da sklad predstavimo v računalniku. Načinov predstavitve je več, odvisno tudi od uporabe določenega programskega jezika. Uporabili

Slika 2



bomo kar BASIC in predstavitev s pomočjo tabele.

Za podatkovno strukturo **"sklad"** je povsem dovolj, da poznamo njen vrh, kajti tja vstavljamo in od tam brišemo podatke, ter da poznamo razvrstitev elementov. Če uporabimo **linearno predstavitev** ali predstavitev s tabelo, je vrstni red določen kar z indeksom elementa v tabeli. Element, ki je na vrhu sklada, ima najvišji indeks, njegov predhodnik indeks za eno manjši, ... V definiciji sklada nikjer nismo omejili števila elementov, ki so naenkrat lahko v skladu. Ker pa si moramo za tabelo rezervirati prostor, bomo z VELIKOST označili maksimalno število elementov, ki jih istočasno lahko vodimo v skladu. Kot smo rekli, bomo sklad predstavili s tabelo, ki ji bomo rekli SKLAD. Potrebujemo še spremenljivko, ki označuje vrh sklada. To naj bo kar VRH.

Če smo v sklad zaporedoma vstavili števila 3, 1, 6 in 8 in imamo v skladu prostor za 9 elementov, bo sklad v tem trenutku predstavljen takole:

```
VELIKOST = 9
VRH  = 4

SKLAD(1) = 3        SKLAD(4) = 8    SKLAD(7) = ...
SKLAD(2) = 1        SKLAD(5) = ...  SKLAD(8) = ...
SKLAD(3) = 6        SKLAD(6) = ...  SKLAD(9) = ...
```

Sestavimo sedaj podprogram, s pomočjo katerih izvajamo potrebne operacije.

```
10  REM
20  REM pripravi sklad velikosti VELIKOST z vrhom VRH
30  REM
40  VELIKOST = ...
50  DIM SKLAD(VELIKOST)
60  VRH = 0
70  RETURN
```

```
100 REM
110 REM če je sklad prazen, dobi spremenljivka PRAZEN
120 REM vrednost 1, drugače 0
130 REM
140 PRAZEN = 0
150 IF VRH = 0 THEN PRAZEN = 1
160 RETURN

200 REM
210 REM vstavi podatek ELEMENT v sklad
220 REM
230 IF VRH < VELIKOST THEN GOTO 260
240   REM sklad je poln - primerno ukrepamo
250   ...
260 REM sklad ni poln
270 VRH = VRH + 1
280 SKLAD(VRH) = ELEMENT
290 RETURN

300 REM
310 REM briše podatek iz sklada in ga shrani v ELEMENT
320 REM
330 IF VRH = 0 THEN ... : REM napaka, primerno ukrepamo
340 ELEMENT = SKLAD(VRH)
350 VRH = VRH - 1
360 RETURN
```

Zadnji podprogram vsebuje v sebi dve operaciji: ODSTRANI, ki iz sklada naredi nov sklad, in VRH, ki vrne vrhnji element v skladu. S ... smo označili mesta, kjer vpišemo ukaze, ki ustrezajo uporabi sklada v programu. Tako lahko prekinemo izvajanje programa in sporočimo napako, lahko pokličemo ustrezni podprogram, ... Odvisno pač od situacije, v kateri uporabljamo sklad. V nekaterih primerih bo to znak, da je v programu napaka, drugič spet le znamenje, da program potrebuje večji sklad.

V začetku smo omenili, da je sklad le ena od osnovnih podatkovnih struktur. Druge, ki jih prav tako pogosto uporabljamo v računalništvu, so še: drevo, vrsta, kopica, graf, gozd in še mnoge druge. V naslednjih številkah PRESEKA si bomo ogledali še nekatere. Do takrat pa nekaj nalog, ki so objavljene na strani 333.

Matija Lokar