PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **28** (2000/2001)
Številka 1
Strani 42–47

Martin Juvan:

SPREMENLJIVO ŠTEVILO PARAMETROV

Ključne besede: računalništvo, programiranje, podprogrami, parametri, heksadecimalni zapis.

Elektronska verzija: http://www.presek.si/28/1430-Juvan.pdf

SPREMENLJIVO ŠTEVILO PARAMETROV

Praktično vsi programski jeziki poznajo podprograme, ki sprejmejo spremenljivo število parametrov. Predvsem podprogrami za branje in izpisovanje imajo zelo pogosto tako obliko, na primer *read* in *write* v pascalu ali pa *scanf* in *printf* v C-ju. Veliko jezikov pozna tudi konstrukte, ki programerju omogočajo, da sam napiše podprograme, ki sprejmejo spremenljivo število parametrov. V C-ju, na primer, take funkcije sprogramiramo s pomočjo ukazov iz standardne knjižnice `stdarg.h`. V tem prispevku si bomo ogledali, kakšne možnosti za sestavljanje ukazov s spremenljivim številom parametrov nam nudi MSWLogo, različica loga, prilagojena za okolja Windows (glej `http://vlado.fmf.uni-lj.si/educa/logo`).

Logo pozna veliko vgrajenih ukazov, ki jih lahko pokličemo z različnim številom parametrov. Med pomembnejšimi takimi ukazi so ukazi za izpisovanje `PRINT`, `SHOW` in `TYPE` ter ukazi za sestavljanje besed oziroma seznamov `WORD`, `LIST` in `SENTENCE`. Vsak ukaz v logu ima določeno *privzeto* število parametrov, to je tisto število parametrov, s katerim lahko pokličemo ukaz, ne da bi klic morali obdati z oklepaji. Na primer, pravkar našteti ukazi za izpisovanje privzeto sprejmejo en sam parameter. Tako s klicem

```
PRINT "Presek
```

izpišemo besedo `Presek`. Če pa poskusimo na enak način izpisati več besed, na primer

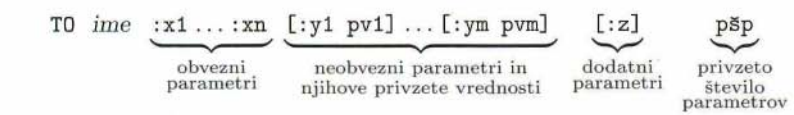
```
PRINT "list "za "mlade "matematike "fizike "...,
```

se izpiše le beseda `list`, tolmač pa nam sporoči, da ne ve, kaj storiti z ostalimi besedami. Seveda, pri zgornjem klicu ukaz `PRINT` vzame le en parameter. Če ga želimo uporabiti z več parametri, moramo to logu posebej povedati. To storimo tako, da klic ukaza skupaj z vsemi parametri obdamo z okroglimi oklepaji. Na primer, z zapisom

```
(PRINT "list "za "mlade "matematike "fizike "...)
```

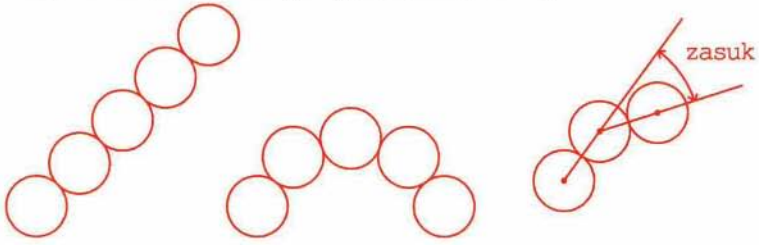
dosežemo, da ukaz `PRINT` kot parametre prejme vseh šest besed in jih tudi izpiše.

V logu za sestavljanje lastnih ukazov uporabljamo vgrajeni ukaz `TO`. Splošna oblika tega ukaza v MSWLogu je



Obvezne parametre moramo navesti pri vsakem klicu ukaza. Število obveznih parametrov je torej najmanjše število parametrov, s katerimi lahko pokličemo ukaz. Neobveznih parametrov pri klicu ni nujno navesti. Če jih (nekaj) izpustimo, bo ukaz namesto njih uporabil privzete vrednosti. Del z dodatnimi parametri nam omogoča, da ukaz pokličemo s poljubno veliko parametri. Zadnji del ukaza `TO` je število, ki določi privzeto število parametrov za ukaz. Če tega števila ne navedemo, je privzeto število parametrov enako številu obveznih parametrov.

Poglejmo nekaj primerov. Najprej si bomo ogledali ukaz, ki nariše zaporedje dotikajočih se krogov (glej spodnjo sliko).



Ukaz bo imel dva obvezna parametra, število krogov `n` in polmer krogov `r`. Da bo pri uporabi več svobode, bomo dodali še neobvezni parameter `zasuk`, s katerim določimo, za koliko je premica skozi središči trenutnega in naslednjega kroga zasukana glede na zveznico središč prejšnjega in trenutnega kroga (glej desni del zgornje slike). Privzeta vrednost za `zasuk` bo 0 stopinj; tedaj vsa središča krogov ležijo na skupni premici. Ukaz izgleda takole:

```
TO krogi :n :r [:zasuk 0]
  LOCALMAKE "pero PENDOWNP    ; Zapomnimo si stanje peresa.
  REPEAT :n [
    PENDOWN CIRCLE :r         ; Spustimo pero in narišemo krog.
    RT :zasuk                 ; Opravimo zasuk.
    PENUP FD 2 * :r           ; Pomik v središče naslednjega kroga.
  ]
  IF :pero [PENDOWN]          ; Pero vrnemo v začetno stanje.
END
```

Levi del zgornje slike narišemo na sredino zaslona z ukazi

```
CS RT 45 PU BK 200 PD krogi 5 50,
```

srednji del pa dobimo s klicem

```
CS (krogi 5 50 36).
```

Drugi primer bo bolj računski. Sestavili bomo ukaz, ki bo kot obvezni parameter dobil naravno število n > 0, vrnil pa bo niz, ki bo predstavljal šestnajstiški zapis števila n. Šestnajstiškemu zapisu pravimo tudi *heksadecimalni* zapis. Pri njem poleg običajnih števk $0, 1, \ldots, 9$ kot števke uporabljamo še črke A (=10), B (=11), C (=12), D (=13), E (=14) in F (=15). Tako je 7D0 šestnajstiški zapis števila 2000, saj je

$$7 \cdot 16^2 + 13 \cdot 16 + 0 \cdot 1 = 1792 + 208 + 0 = 2000\,.$$

Ukaz bo imel še neobvezni parameter d. Ta določi, koliko znakov ima beseda, ki jo vrne ukaz. Njegova privzeta vrednost bo ravno število števk, ki jih potrebujemo za šestnajstiški zapis števila n. To vrednost dobimo tako, da celemu delu šestnajstiškega logaritma števila n prištejemo 1. Ker logo ne pozna logaritmov z osnovo 16, si bomo pri računanju pomagali z desetiškim logaritmom. V MSWLogu ga vrne funkcija LOG10. Če bo vrednost parametra d večja od števila potrebnih števk, bomo rezultat na začetku z ničlami dopolnili do želene dolžine. Če pa bo vrednost parametra d manjša od števila števk, bo v rezultatu manjkalo nekaj vodilnih števk. Tule je koda ukaza:

```
TO sestnajst :n [:d 1 + INT ((LOG10 :n) / (LOG10 16))]
; Vrne besedo, ki predstavlja zadnjih d števk šestnajstiškega
; zapisa števila n. Če je zapis krajši, je dopolnjen z ničlami.
  LOCALMAKE "stevke [0 1 2 3 4 5 6 7 8 9 A B C D E F]
  LOCALMAKE "beseda "
  REPEAT :d [
    MAKE "beseda WORD (ITEM (1 + REMAINDER :n 16) :stevke) :beseda
    MAKE "n INT :n / 16
  ]
  OUTPUT :beseda
END
```

Klic `sestnajst 2000` tako vrne besedo `7D0`, klica `(sestnajst 2000 5)` in `(sestnajst 2000 2)` pa besedi `007D0` oziroma `D0`. Kot vidimo pri gornjem ukazu, lahko za privzete vrednosti neobveznih parametrov določimo poljubne izraze. V njih lahko uporabljamo vrednosti obveznih parametrov, pa tudi vrednosti predtem že navedenih neobveznih parametrov.

Kot naslednji primer si oglejmo ukaz, s katerim bomo lahko narisali krožne kolobarje, kot so prikazani na naslednji sliki.

Ukaz bo imel tri parametre, enega obveznega in dva neobvezna. Dodatno bomo še določili, da bo privzeto število parametrov enako 2. Prvi parameter bo polmer zunanjega kroga, drugi parameter bo polmer notranjega kroga, tretji parameter pa barva v obliki RGB (seznam treh števil med 0 in 255), s katero bo obarvan kolobar. Privzeta vrednost za polmer notranjega kroga bo polovica polmera zunanjega kroga, privzeta barva pa bo kar barva peresa ob klicu.

```
TO kolobar :r1 [:r2 :r1 / 2] [:barva PENCOLOR] 2
  ; Zapomnimo si stanje peresa ter barvi peresa in zapolnjevanja.
  LOCALMAKE "pero PENDOWNP
  LOCALMAKE "b_pero PENCOLOR
  LOCALMAKE "b_poln FLOODCOLOR
  ; Nastavimo izbrano barvo peresa in barvo zapolnjevanja.
  SETPENCOLOR :barva  SETFLOODCOLOR :barva
  ; Spustimo pero in narišemo oba kroga.
  PD CIRCLE :r1  CIRCLE :r2
  ; Premik na sredino med oba kroga, pobarvamo in se vrnemo.
  PU FD (:r1 + :r2) / 2 FILL BK (:r1 + :r2) /2
  ; Vzpostavimo začetno stanje peresa in obeh barv.
  IF :pero [PENDOWN]
  SETPENCOLOR :b_pero
  SETFLOODCOLOR :b_poln
END
```

Prvi kolobar lahko dobimo s klicem

```
(kolobar 100).
```

Klic moramo obdati z oklepaji, sicer tolmač javi, da manjka parameter. Drugi kolobar narišemo s klicem

```
kolobar 100 90.
```

Je precej tanjši od prvega, saj je polmer notranjega kroga kar $\frac{9}{10}$ polmera zunanjega kroga, pri prvem kolobarju pa je bil ta polmer $\frac{1}{2}$ zunanjega.

Zadnji kolobar je debelejši in je dobljen s klicem

```
(kolobar 100 25 [195 195 195]).
```

Seznam `[195 195 195]` v zapisu RGB določa bled odtenek sivine (približno 20% črne barve; črno barvo dobimo z `[0 0 0]`, belo pa z `[255 255 255]`).

Nazadnje si oglejmo še ukaz, ki bo sprejel tudi dodatne parametre. Ukaz bo vzel zaporedje dveh ali več točk (vsaka točka bo podana kot par, torej seznam, števil) in narisal lomljeno črto, sestavljeno iz daljic, pri čemer bo prva daljica potekala od prve do druge točke, druga od druge do tretje točke, itn.

```
TO lomljenka :tc1 :tc2 [:tc]
  ; Zapomnimo si začetni položaj in stanje peresa.
  LOCALMAKE "zacpol POS
  LOCALMAKE "pero PENDOWNP
  PU SETPOS :tc1 PD            ; Premik v prvo točko.
  SETPOS :tc2                  ; Daljica do druge točke.
  REPEAT COUNT :tc [           ; Zanka po dodatnih točkah.
    SETPOS ITEM REPCOUNT :tc
  ]
  PU SETPOS :zacpol            ; Vrnemo se na začetni položaj.
  IF :pero [PENDOWN]           ; Vzpostavimo začetno stanje peresa.
END
```

Ukaz ima dva obvezna parametra, neobveznih parametrov nima, dopušča pa dodatne parametre. Ko ukaz pokličemo, moramo navesti vsaj dva parametra. Tolmač poskrbi, da se dodatni parametri združijo v seznam in ta seznam je začetna vrednost spremenljivke `tc`. Če ukaz pokličemo le z dvema točkama, bo `tc` prazen seznam.

Ukaz lahko uporabimo za izpisovanje z "velikimi" črkami. Na primer, zgornji napis prikažemo na sredini zaslona z naslednjim zaporedjem klicev:

```
(lomljenka [-330 120] [-330 -120] [-210 -120])
(lomljenka [-150 -120] [-30 -120] [-30 120] [-150 120] [-150 -120])
(lomljenka [90 0] [150 0] [150 -120] [30 -120] [30 120] [150 120])
(lomljenka [210 -120] [330 -120] [330 120] [210 120] [210 -120])
```

V primerih smo uporabili tudi nekaj manj znanih logovih ukazov, ki jih nismo posebej razložili. Kratke razlage njihovega delovanja so zbrane v naslednjem seznamu:

`LOCALMAKE` je ukaz, ki ustvari lokalno spremenljivko in ji hkrati še priredi začetno vrednost. Uporabljamo ga kot krajši zapis kombinacije ukazov `LOCAL` in `MAKE`.

`PENDOWNP` je ukaz, ki vrne `true`, če je pero spuščeno (pri premikanju želva pušča sled), in `false` sicer. V primerih smo z njegovo uporabo poskrbeli, da je bilo stanje peresa po koncu klica ukaza enako kot pred klicem.

`INT` je ukaz, ki vrne celi del števila ("odreže decimalke"). Tako klic `INT 3.5` vrne `3`, klic `INT -3.5` pa `-3`.

`REMAINDER` je ukaz, ki vzame dve celi števili in vrne ostanek pri celoštevilskem deljenju prvega števila z drugim. Predznak ostanka je enak predznaku prvega števila (to za nas sicer ni bilo pomembno, saj smo ukaz uporabljali le za pozitivna števila). V nekaterih drugih jezikih se enakovredna operacija imenuje *mod*.

`FLOODCOLOR` je ukaz, ki vrne trenutno veljavno barvo zapolnjevanja. To barvo lahko spremenimo z ukazom `SETFLOODCOLOR`. Zapolnjevanje ("barvanje") opravimo z ukazom `FILL`. Novejše izvedbe MSWLoga poznajo dve različici ukaza `FILL`. Obe začneta barvati na mestu, kjer se nahaja želva. Pri klicu `FILL`, ta je enakovreden klicu `(FILL "false)`, se barva "razlije" le po strnjenem območju, ki je obarvano enako kot pika, na kateri se je barvanje začelo. Pri klicu `(FILL "true)` pa se barvanje širi toliko časa, dokler ga ne zaustavijo pike, obarvane s trenutno barvo peresa. Na primer, zaporedje ukazov

```
CS SETPC [0 0 0] CIRCLE 100 SETPC [255 0 0] FILL
```

nariše črn krog in ga obarva, zaporedje

```
CS SETPC [0 0 0] CIRCLE 100 SETPC [255 0 0] (FILL "true)
```

pa tudi nariše krog, nato pa obarva ves zaslon in ne samo kroga. Seveda, krog je narisan s črno barvo, pred klicem ukaza `FILL` pa smo barvo peresa spremenili na rdeče, tako da črno narisan krog barvanja ne ustavi.

`REPCOUNT` je ukaz, ki nam znotraj zanke `REPEAT` pove, v kateri ponovitvi zanke smo. Ta ukaz nam nadomešča števec, ki ga običajno uporabljamo pri zankah *for* v drugih programskih jezikih. Mimogrede, tudi MSWLogo pozna ukaz `FOR`.

Martin Juvan