KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

Michal Krupka
krupka.inf.upol.cz
michal.krupka@upol.cz
17. listopadu 12, 771 46 Olomouc

Paradigmata programování 2 ◇ poznámky k přednášce

11. Proměnné a podprogramy

verze z 27. května 2024

1 Proměnné

Snadno zjistíme, že náš zásobníkový stroj zatím neumí řešit všechny problémy řešitelné jinými programovacími jazyky. Je to dáno tím, že umí pracovat pouze s prvními třemi prvky na datovém zásobníku. Například úlohu 2. na konci tohoto textu v něm vyřešit nelze (zkuste to).

Jeden z možných způsobů, jak to vyřešit, by bylo zavést nový pomocný zásobník a slova na odložení prvku datového zásobníku na něj a zpět. To by také umožnilo definovat některá vestavěná slova jako uživatelská (např. slovo `:rot`). Zájemci si mohou zkusit tento způsob vytvoření plnohodnotného programovacího jazyka z jazyka z minulé přednášky implementovat. Je to velmi snadné. My se ale pustíme jiným směrem a rozšíříme náš jazyk o možnost používání **proměnných**.

Roli proměnných v našem jazyce budou hrát symboly Lispu, které nejsou klíči (jak víme, klíče jsou rezervovány na slova), i když v principu bude možné jako proměnnou použít libovolnou hodnotu. Vytvoříme nový zásobník (slovník), na který budeme ukládat informace o hodnotách proměnných a definujeme nové slovo, které k dané proměnné zjistí její hodnotu. Kromě toho definujeme dvě základní slova na editaci nového zásobníku. Prvky zásobníku budou páry `(`*`proměnná . hodnota`*`)`, kterým budeme říkat *vazby*. Samotnému zásobníku budeme říkat *zásobník vazeb* a budeme ho značit *`bnd`*.

Ke zjištění hodnoty dané proměnné slouží slovo `:val`, k vytváření a rušení vazeb slova `:bind` a `:unbind`.

Slova pro proměnné

`:val`	odstraní proměnnou z vrcholu zásobníku *`rslt`* a dá na něj její hodnotu nalezenou v zásobníku *`bnd`*	(*`var`* – *`val`*)
`:bind`	vytvoří novou vazbu a dá ji na zásobník *`bnd`*	(*`val var`* –)
`:unbind`	odstraní vazbu z vrcholu zásobníku *`bnd`*	(–)

Poznámky

- Vazby se mohou překrývat (nejstarší způsob implementace proměnných s možností překrývání).
- Důsledkem této implementace je, že vazby proměnných jsou **dynamické**.

2 Doplnění interpretu o proměnné

Zásobník vazeb symbolů bude v proměnné `*bnd*`. Rovnou do něj uložíme vazbu symbolu `pi` na číslo:

```
(defvar *bnd*)
(setf *bnd* `((pi . ,pi)))
```

Zásobník funguje jako slovník, můžeme na něj tedy používat funkci `dict-find` a makro `dict-push` definované dříve pro zásobník slov.

Definice vestavěných slov `:bind`, `:unbind` a `:val`:

```
(defprim :bind
  (dict-push (pop *rslt*) (pop *rslt*) *bnd*))

(defprim :unbind
  (pop *bnd*))

(defprim :val
  (push (dict-find (pop *rslt*) *bnd*)
        *rslt*))
```

Proměnné nyní můžeme otestovat třeba na tomto kódu:

```
(execute 3 'x :bind 4 'y :bind 'x :val 'y :val :+ :unbind :unbind)
```

nebo na novém slově, které slouží k výpočtu obsahu kruhu daného poloměru:

```
(execute :def :carea :dup 'pi :val :* :*)
(execute 10 :carea)
```

Kód můžete krokovat, jak jsem vysvětlil minule. Můžete si také upravit funkci `execute`, aby vytvářela novou vazbu na proměnnou `*bnd*`. Pak uvidíte, jak se zásobník v průběhu vykonávání programu mění.

3 Podprogramy

Každý větší program, je vhodné účelně rozdělit na části. Jednak kvůli snazší orientaci a jednak proto, že některé úseky kódu je třeba používat z různých míst programu. V jazycích nižší úrovně, mezi které lze řadit i jazyk našeho zásobníkového stroje, úlohu vedlejších částí programu hrají *podprogramy*.

Jde o úseky kódu, na které může program kdykoli přejít, a po jejichž skončení se vrátí zpět na původní místo v kódu. Z podprogramů je možné před jejich skončením přecházet na další podprogramy; toto vnořování může být v principu neomezené a závisí jen na dostupné paměti. Aby bylo u každého podprogramu známo, kam se má po jeho skončení vykonávání instrukcí vrátit, je třeba, aby si toto místo program někde zapamatoval. Vzhledem k tomu, že podprogramy se mohou vzájemně volat a ukončují se v opačném pořadí, než v jakém byly volány, je vhodné k tomuto účelu použít pomocný zásobník.

Kromě popsané funkčnosti nejsou podprogramy vázány ničím dalším. Především nedělají to, co známe z procedur a funkcí vyšších programovacích jazyků: nepřijímají žádné argumenty a nevracejí hodnoty. Místo toho pracují přímo se zásobníky.

Podprogram budeme reprezentovat seznamem slov a dalších hodnot, tedy stejně, jako celý program. Pokud se takový seznam ocitne na vrcholu programového zásobníku, je podle pravidel stanovených minule v dalším kroku přesunut na vrchol zásobníku datového.

Poznámka: na vrcholu zásobníku není uložen celý seznam, ale jen odkaz (ukazatel) na jeho první tečkový pár. Tuto skutečnost před námi Common Lisp skrývá, ale je zřejmé, že to nemůže být jinak. Všechny prvky libovolného seznamu mají stejnou velikost a jsou to buď přímo jednoduchá data (malá čísla), nebo odkazy na data umístěná jinde. Paměť našeho zásobníkového stroje si tedy můžeme zhruba představit jako lineární úložiště obsahující seznamy.

Při spuštění podprogramu (slovem `:clsub`) se programový zásobník nahradí podprogramem, který se tedy v dalším kroku začne vykonávat. Původní obsah programového zásobníku se zapamatuje: uloží se na nový *návratový zásobník*, který budeme označovat zkratkou *ret*. Jak jsem uvedl v předchozí poznámce, ve skutečnosti se tam uloží jen odkaz na seznam tvořící původní programový zásobník (přesně řečeno jeho první pár); samotný seznam zůstane na místě. Podle stejného principu nový programový zásobník vznikne tak, že se do proměnné `*exec*` uloží odkaz na seznam tvořící podprogram.

V okamžiku, kdy podprogram skončí, je třeba programový zásobník obnovit do původního stavu. O to se musí postarat podprogram sám použitím k tomuto účelu určeného slova (`:ret`).

Uživatelská slova

Jelikož máme k dispozici podprogramy, nemusíme už definovat nová uživatelská slova. Pokud to nebude vyloženě požadováno (třeba v zadání úlohy), nebudeme už uživatelská slova definovat a slovo `:def` přestaneme používat.

K práci s podprogramy tedy potřebujeme dvě nová vestavěná slova.

Slova pro podprogramy

`:clsub`	odstraní podprogram z vrcholu zásobníku *rslt* a nahradí jím programový zásobník jeho původní obsah uloží na návratový zásobník
`:ret`	vypustí z návratového zásobníku vrchol a nahradí jím programový zásobník

4 Implementace a použití podprogramů

Návratový zásobník uložíme do dynamické proměnné `*ret*`, kterou jsme definovali už posledně. Funkce `execute` vytváří na tuto proměnnou novou vazbu, a to jednak kvůli snadnějšímu ladění a jednak proto, aby v proměnné po ukončení funkce `execute` nepřekážely případné zapomenuté hodnoty.

Definice slov `:clsub` a `:ret`:

```
(defprim :clsub
  (push *exec* *ret*)
  (setf *exec* (pop *rslt*)))

(defprim :ret
  (setf *exec* (pop *ret*)))
```

Podprogramy je vhodné ukládat do proměnných a volat je použitím slov `:val` a `:clsub`. Každý podprogram musí na konci použít slovo `:ret`, jinak se běh nevrátí tam, odkud byl podprogram volán.

Definice nových podprogramů a jejich použití najdete ve zdrojovém kódu k přednášce.

5 Podprogramy jako procedury

Další informace o podprogramech, kterou si řekneme, se týká **parametrů a návratových hodnot**. Podprogramy, jak jsme je zde zavedli, představují pouze část kódu, kterou lze někam uložit (na zásobník, do proměnné) a později spustit. Nejde o procedury v běžném slova smyslu, protože neimplementují parametry a návratovou hodnotu.

Podprogramu budeme říkat **procedura**, pokud splňuje následující podmínky:

1. Podprogram odebere z datového zásobníku dopředu stanovený počet prvků (*argumentů*).
2. Po skončení výkonu podprogramu bude na datovém zásobníku jedna nová hodnota (*návratová hodnota*).
3. Po skončení výkonu podprogramu bude zásobník vazeb ve stejném stavu jako před spuštěním.

Podprogram můžeme jako proceduru napsat následujícím způsobem:

```
(y :bind
 x :bind
     tělo používající x a y
   :unbind
   :unbind
 :ret)
```

(Při použití ve funkci `execute` musíme symboly `x` a `y` kvotovat, protože jinak by Lisp v programu použil jejich hodnoty, nikoli symboly samotné.)

Tento podprogram emuluje proceduru se dvěma parametry, `x` a `y`. Výhodou takto napsaného podprogramu je, že v jeho těle můžeme s parametry pracovat podobně jako u klasické procedury.

Například při vykonávání programu

```
1 2 podprogram :clsub
```

kde *podprogram* je podprogram napsaný výše, se hodnoty `1` a `2` nejprve uloží na datový zásobník, pak se z něj odeberou a navážou na symboly `x` a `y` (v tomto pořadí!). Vazby symbolu `x` na `1` a symbolu `y` na `2` budou platné po celou dobu vykonávání podprogramu a pak zaniknou.

Aby se podprogram skutečně choval jako procedura, musí ještě na datovém zásobníku po skončení nechat právě jednu hodnotu, která bude představovat návratovou hodnotu procedury.

Otázky a úlohy na cvičení

Pokud je v úloze uvedeno, že máte definovat slovo, definujte ho pomocí slova `:def` z minulé přednášky. Pokud máte definovat proceduru, udělejte to bez slova `:def`.

1. K testování některých z následujících příkladů se vám může hodit pomocné slovo `:print`, které vytiskne vrchol datového zásobníku (ale nevymaže).

2. Definujte slovo `:avgn` na výpočet aritmetického průměru n čísel. n je na vrcholu zásobníku, čísla jsou pod ním:

```
(a1 ... an n -- průměr čísel a1 ... an)
```

   Můžete přitom použít slovo `:+n` z úlohy k minulé přednášce.

3. Definujte a uložte do proměnné `add-1` podprogram, který k danému číslu přičte jedničku. Program

```
5 add-1 :val :clsub
```

   tedy nechá na datovém zásobníku číslo `6`.

4. Definujte vestavěná slova `:<` a `:>` na porovnávání čísel. Pak čistě v našem zásobníkovém jazyce definujte podprogram na výpočet hodnoty funkce sgn (znaménko) a uložte jej do proměnné `sgn`:

```
> (execute 5 'sgn :val :clsub)

1

> (execute 0 'sgn :val :clsub)

0

> (execute -2 'sgn :val :clsub)

-1
```