Uma Aula Prática sobre LISP

Siang Wun Song - Universidade de São Paulo - IME/USP

MAC 5710 - Estruturas de Dados - 2008

Linguagem LISP

- Surgiu no final dos anos 50 - John McCarthy (MIT).
- Linguagem para processamento simbólico.
- Muito usada em aplicações em Inteligência Artificial. Outras aplicações incluem AutoCad, Editor Emacs, etc.
- Linguagem funcional - LISP puro não possui atribuições (livre de efeitos colaterais).
- Lisp não é compliado (código fonte para código objeto de máquina), mas sim interpretado por meio de um interpretador.
- Interpretador é implementado usando estruturas de listas encadeadas.
- Coletor de lixo faz parte do sistema.

MuLISP

Há muitos dialetos de LISP:

- Mac Lisp
- Franz Lisp
- Common Lisp
- Golden Lisp
- MuLisp, etc.

Uma página contendo vários sistemas LISP:
http://magnum.ime.uerj.br/∼mpmelo/IME4701/2002-2.htm

Linguagem Funcional

- Aplicação de função a seus argumentos, com a possibildiade de composição funcional.
- Uso intenso de recursão.
- LISP significa “LISt Processor”
- Dijkstra brincava dizendo: LISP é a maneira mais elegante de disperdiçar recursos :-)
- Outra brincadeira: LISP significa
  Lot of **I**rritating **S**tupid **Pa**renthesis.

MuLISP5.1

MuLISP foi desenvolvido nos anos 80 para o sistema DOS. A partir de MuLISP a empresa Soft Warehouse de Honolulu desenvolveu o MuMath cujo sucessor é o sistema Derive, um sistema de álgebra computacional.

MuLISP é bem compacto. O código ocupa apenas 38 Kbytes.

Possui aritmética de precisão “ilimitada” (como o nosso programa 1 do curso).

Demonstração MuLISP

No prompt do sistema DOS, tecle mulisp.

O sinal de prompt do MuLISP é o sinal de dólar $

Depois do prompt $ pode-se entrar uma função seguida de argumentos, se houver.

O interpretador MuLISP vai avaliar a função e retornar o resultado da avaliação. Exemplos:

- $ 5
  5
  A função constante 5 vai devolver o número 5.
- $ (+ 3 5)
  8
  O primeiro elemento da lista “+” é interpretado como uma função (soma), os demais elementos (3 e 5) são argumentos da função. O MuLISP retorna o valor da soma.

Demonstração MuLISP

- $ '(+ 3 5) ou (quote (+ 3 5))
  (+ 3 5)
  O símbolo ' ou a função quote faz com que o argumento em seguida seja tratado como um valor literal que, portanto, não é avaliado.
- $ (+ 3 (- 10 2))
  11
  Composição de funções é possível.
- $ (+ (* 5 2) (- 10 2))
  18
  Mais composições de funções.
- $ (* 123456789 987654321)
  121932631112635269
  Precisão "ilimitada": cada número é armazenado como uma lista ligada de algarismos.
- (system)
  Termina o interpretador MuLISP e retorna-se ao DOS.

Demonstração MuLISP

- $ (zerop 9)
  nil
  A função zerop testa se o argumento é zero. Se sim retorna T caso contrário retorna nil.
- $ (zerop (- 100 (+ 80 20)))
  T
- $ (sub1 60)
  59
  A função sub1 subtrai um.
- $ (defun cubo (n) (* n n n))
  cubo
  Defun serve para definir uma função. Após a definição o sistema retorna o nome da função. De agora em diante, pode-se usar a função definida.
- $ (cubo 2)
  8

Demonstração MuLISP

- $ (defun fatorial (n) (cond
  ((zerop n) 1)
  (T (* n (fatorial (sub1 n))))
)
)
  fatorial
  Cond é uma condicional. Segindo Cond vêm pares de funções. Avalia-se o primeiro elemento de cada par, sucessivamente, até se chegar a um valor não-nil. Entao o valor de Cond é o valor do segundo elemento do par.
- $ (fatorial 5)
  120
  Experimente chamar fatorial com número bem grande.
- $ (rds doctor.lsp)
  doctor
  A função rds lê o programa escrito em MuLISP especificado pelo argumento e inicia a avaliação.

Próximo termo de uma sequência

```
(defun nextterm (x) (list
     (quote "O proximo termo da sequencia ") x (quote  = ) (next x))
)
(defun next (x) (nthterm (add1 (length x))
                         (encode x))
)
(defun encode (x) (cond
                       ( (null (cdr x))  (cons (car x) nil))
                       ( (p1 x)          (cons (car x) nil))
                       (  T              (cons (car x) (encode (d x))))
                  )
)
(defun p1 (x)     (cond
                       ( (null (cdr (cdr x)))          (equal (car x) (car (cdr x))))
                       ( (equal (car x) (car (cdr x))) (p1 (cdr x)))
                       ( T                             nil)
                  )
)
(defun d (x)      (cond
                       ( (null (cdr x))       nil)
                       ( T                    (cons (- (car (cdr x)) (car x))
                                                    (d (cdr x))))
                  )
)
(defun nthterm (n e) (cond
                       ( (equal n 1)     (car e))
                       ( T               (nthterm (sub1 n) (st e)))
                     )
)
(defun st (e)       (cond
                       ( (null (cdr e))   (cons (car e) nil))
                       ( T                (cons (+ (car e) (car (cdr e)))
                                                (st (cdr e))))
                     )
)
```

Este programa recebe uma sequência de números e tenta descobrir o próximo. Por ex: $ (nextterm (quote (1 2 3 4 5)))
vai produzir o número 6. Experimente outras sequências.