3. Métodos e Técnicas de Programação Funcional: Linguagem LISP

3.1. Aspectos Gerais

O que é LISP?

LISP é uma linguagem de programação funcional. Foi inventado por J. McCarthy em 1959.

Houve épocas em que havia muitos dialetos de LISP.
Hoje em dia LISP está estandardizado no padrão COMMON LISP.

Há aplicações de LISP nos domínios do
processamento simbólico e de conhecimento (IA),
processamento numérico (MACLISP),
e na confecção de programas muito difundidos como editores (EMACS) e CAD (AUTOCAD).

O texto de referência padrão da linguagem é: Guy L. Steele Jr.: Common Lisp - The Language. Digital Press.
1. edition 1984, 465 pages.
2. edition 1990, 1032 pages.

Este livro está disponível em HTML via FTP de:

ftp.cs.cmu.edu:/user/ai/lang/lisp/doc/cltl/cltl_ht.tgz and

http://www.cs.cmu.edu:8001/Web/Groups/AI/html/cltl/cltl2.html

http://www.cs.cmu.edu:8001/afs/cs/project/ai-repository/ai/html/cltl/cltl2.html

LISP é rodado em um ambiente interativo.

Você entra com “forms” - conjuntos de expressões - e elas são avaliadas de uma só vez.

Você também pode inspecionar variáveis, chamar funções com argumentos dados e definir suas pr´oprias funções.

No decorrer desta disciplina, nos vamos utilizar o CLISP, uma implementação de COMMON LISP.

CLISP está quase completamente de acordo (99%) com a definição de COMMON LISP e inclui ainda CLOS, um dialeto LISP orientado a objetos.

Common Lisp é

- uma linguagem de programação funcional com usos convencionais e uma linguagem para Inteligência Artificial
- interativo

Programas em Common Lisp são altamente portáveis entre máquinas e sistemas operacionais (there is a standard for the language and the library functions)

Common Lisp provê

- sintaxe clara
- muitos tipos de dados: numbers, strings, arrays, lists, characters, symbols, structures, streams etc.
- tipagem em tempo de execução: o programador geralmente não precisa se preocupar com declarações de tipo, mas ele recebe mensagens de erro caso haja violações de tipo (operações ilegais) .
- Funções genéricas: 88 arithmetic functions for all kinds of numbers (integers, ratios, floating point numbers, complex numbers), 44 search/filter/sort functions para listas, arrays e strings
- gerenciamento de memória automático (garbage collection)
- pacoteamento (packaging) de programas em módulos
- um sistema de objetos, funções genéricas com a possibilidade de combinação de métodos.
- macros: todo programador pode realizar suas próprias extensões da linguagem

A implementação de Common Lisp CLISP provê:

- um interpretador.
- um compilador para executáveis até 5 vezes mais rápidos.
- todos os tipos de dados com tamanho ilimitado (a precisão e o tamanho de uma variável não necessita de ser declarado, o tamanho de listas e arrays altera-se dinamicamente)
- integers de precisão arbitrária, precisão de ponto flutuante ilimitada.

3.2. Tutorial da Linguagem LISP

Baseado em:

Common LISP Hints
Geoffrey J. Gordon
<ggordon@cs.cmu.edu>
Friday, February 5, 1993

3.2.1. Symbols

Um símbolo é somente um string de caracteres.
Alguns exemplos:

```
a
b
c1
foo
bar
baaz-quux-garply
```

Algumas operações com símbolos seguem abaixo. Coisas após o prompt “>” são as que você digita para o interpretador lisp. Tudo após um “;” é um comentário.

```
> (setq a 5)              ;store a number as the value of
a symbol
5
> a                       ;take the value of a symbol
5
> (let ((a 6)) a)         ;bind the value of a symbol
temporarily to 6
6
> a                      ;the value returns to 5 once the
                         ;let is finished
5
```

```
> (+ a 6) ;use the value of a symbol as an argument to a function
11
> b       ;try to take the value of a symbol which has no value
Error: Attempt to take the value of the unbound symbol B
```

Há dois símbolos especiais, t e nil. O valor de t é definido sempre como sendo t. nil é definido como sendo sempre nil.

LISP utiliza t e nil para representar verdadeiro e falso. Exemplo no IF:

```
> (if t 5 6)
5
> (if nil 5 6)
6
> (if 4 5 6)
5
```

O último exemplo é estranho, mas está correto. nil significa falso e qualquer outra coisa verdadeiro. Usamos t somente para clareza.

Símbolos como nil e t são chamados símbolos auto-avaliantes, porque avaliam para si mesmos.

Há toda uma classe de símbolos auto-avaliantes chamados palavras-chave. Qualquer símbolo cujo nome inicia com dois pontos é uma palavra-chave. Exemplos:

```
> :this-is-a-keyword
:THIS-IS-A-KEYWORD
> :so-is-this
:SO-IS-THIS
> :me-too
:ME-TOO
```

3.2.2. Números

Um inteiro é um string de dígitos opcionalmente precedido de um + ou -.

Um real parece com um inteiro, só que possui um ponto decimal e pode opcinalmente ser eescrito em notação científica.

Um racional se parece com dois inteiros com um / entre eles.

LISP suporta números complexos que são escritos `#c(r i)`

Exemplos:

```
 17
-34
+6
3.1415
1.722e-15
#c(1.722e-15 0.75)
```

As funções aritméticas padrão são todas avaliáveis: `+, -, *, /, floor, ceiling, mod, sin, cos, tan, sqrt, exp, expt` **etc**

Todas elas aceitam qualquer número como argumento:

```
> (+ 3 3/4)          ;type contagion
15/4
> (exp 1)            ;e
2.7182817
> (exp 3)            ;e*e*e
20.085537
> (expt 3 4.2)       ;exponent with a base other than e
100.90418

> (+ 5 6 7 (* 8 9 10)) ;the fns +-*/ all accept multiple arguments
```

Não existe limite para o valor absoluto de um inteiro exceto a memória do computador. Evidentemente cálculos com inteiros ou racionais imensos podem ser muito lentos.

3.2.3. Conses - Associações

Um `cons` é somente um registro de dois campos. Os campos são chamados de “car” e “cdr” por razões históricas: na primeira máquina onde LISP foi implementado havia duas instruções assembler CAR e CDR "contents of address register" e "contents of decrement register".

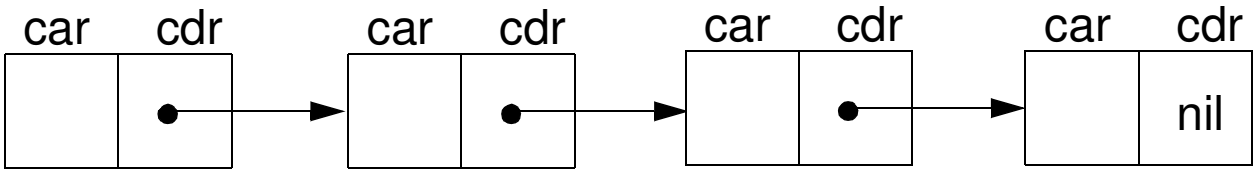
Conses foram implementados utilizando-se esses dois registradores:

```
> (cons 4 5) ;Allocate a cons. Set the car to 4 and the cdr to 5.
(4 . 5)
> (cons (cons 4 5) 6)
((4 . 5) . 6)
> (car (cons 4 5))
4
> (cdr (cons 4 5))
5
```

3.2.4. Listas

Você pode construir muitas estruturas de dados a partir de conses. A mais simples com certeza é a lista encadeada:

- o car de cada cons aponta para um dos elementos da lista e
- o cdr aponta ou para outro cons ou para nil.



Uma lista assim pode ser criada com a função de lista:

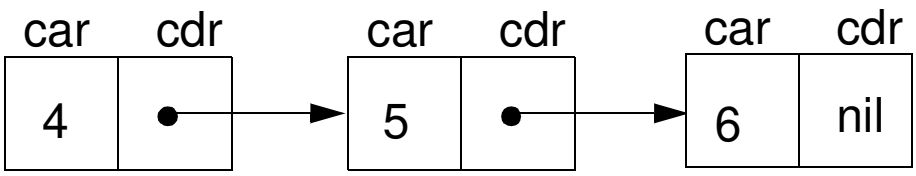
```
> (list 4 5 6)
(4 5 6)
```

Observe que LISP imprime listas de uma forma especial: ele omite alguns dos pontos e parênteses.

A regra é: se o cdr de um cons é nil, lisp não se preocupa em imprimir o ponto ou o nil. Se o cdr de cons A é cons B, então lisp não se preocupa em imprimir o ponto para A nem o parênteses para B:

```
> (cons 4 nil)
(4)
> (cons 4 (cons 5 6))
(4 5 . 6)
> (cons 4 (cons 5 (cons 6 nil)))
(4 5 6)
```

O último exemplo corresponde ao `(list 4 5 6)` anterior. Note que nil corresponde à **lista vazia**.



- O car e cdr de nil são definidos como nil.
- O car de um átomo é o próprio átomo.
- O cdr de um átomo é nil.

Se você armazena uma lista em uma variável, pode fazê-la funcionar como uma pilha:

```
> (setq a nil)
NIL
> (push 4 a)
(4)
> (push 5 a)
(5 4)
> (pop a)
5
> a
(4)
> (pop a)
4
> (pop a)
NIL
> a
NIL
```

3.2.5. Exercícios

3.2.5.1. Desenhe as representações internas de dados para as listas seguintes:
(A 17 -3)
((A 5 C) %)
((A 5 C) (%))
(NIL 6 A)
((A B))
(* (+ 15 (- 6 4)) -3)

3.2.5.2. Qual é o CAR de cada uma das listas do exercício anterior?

3.2.5.3. Qual é o CDR de cada uma das listas do exercício anterior 1?

3.2.5.4. Escreva as declarações necessárias, usando CAR e CDR, para obter os valores seguintes das listas do exercício 1:

(-3)	(-3 -3)
(C %)	(A C %)
(5 %)	(5 (%))
(6 (6))	(6 (6) 6)
((B) A)	(A ((B) B))

3.2.5.5. Defina uma representação conveniente na forma de lista para um conjunto de sobrenomes juntamente com os números de telefones de pessoas. O número de telefone deve permitir a inclusão de códigos de DDD e DDI para números não locais. Como resolveria o caso para pessoas que estivessem na lista, mas não tivessem telefone ?