INTRODUÇÃO

A partir dos anos 50, houve um crescimento significativo ao interesse pela inteligência artificial. Nesse contexto, várias áreas que iam de exatas até humanas, aonde matemáticos, psicólogos, médicos entre outros, buscavam algo que satisfizesse um interesse que tinham em comum: o processamento de dados através de símbolos organizados em listas. Uma vez que na época a computação era baseada em dados numéricos com um armazenamento matricial, ficava difícil a construção de tarefas específicas que necessitavam de uma construção mais clara e sistematizada, utilizando símbolos. Iniciaria então um projeto computacional que tornaria possível a construção de códigos que se aproximassem à linguagem matemática e que fossem interpretados pelo computador em forma de listas encadeadas.

Lisp (List Processing, traduzido do inglês, processamento de listas) foi uma linguagem criada em 1960 por John McCarthy e que satisfazia as necessidades dos entusiastas pela inteligência artificial em códigos, chegando a ser a mais utilizada na área (McCarthy et al. 1962; Foster 1967). Segundo Howard Gardner em A Nova Ciência Da Mente, McCarthy dizia que "O caminho para tornar as máquinas inteligentes deve ser reduzido a um conjunto de relações lógicas e axiomas que podem ser expressos com precisão em termos matemáticos", característica essa que foi adotada para a construção de Lisp.

**1. Valores e Tipos**

**2. Declarações:**

2.1 **Constantes**

2.2 **Variáveis**

2.3 **Tipos**

2.4 **Procedimentos**

**3. Variáveis**

Em Common Lisp, as variáveis são mais conhecidas como átomos, que são partes menores constituintes de uma expressão em Lisp, estes podem ser divididos em símbolos e constantes. “Átomos simbólicos podem ser compostos por letras, números e [...] caracteres alfanuméricos” (LUGER, 2004, p. 620). Um símbolo pode ser associado a um valor assemelhando-se a uma variável, em termos de linhagens imperativas. Uma variável em Lisp tem um nome representado por qualquer símbolo agregando um valor, lista ou átomo.

Lisp é executado em um ambiente completamente imperativo, por conta disso, o programador pode inspecionar as variáveis.

Nesta linguagem de programação não há declaração de variáveis, isso ocorre dinamicamente quando é realizada a primeira referência dela durante a execução do código. O que leva ao seguinte ponto: uma variável pode ser utilizada de vários modos durante a execução, desde que seu uso atual termine, caso o contrário haverá erro durante a execução. Vale ressaltar que o uso do nome de funções como variáveis pode ocasiona uma falta de legibilidade.

As variáveis locais iniciam-se com NIL (vazio), e sua existência é apenas durante a execução.

O escopo padrão das variáveis é estático, mas quando uma variável é declarada especial, ela torna-se automaticamente dinâmica. É característica da linguagem Lisp que todas as variáveis especiais, geralmente globais, tenham tamanho ilimitado, a precisão do tamanho não precisa ser declarada, pois o tamanho altera-se dinamicamente.

**4. Expressões e Atribuições**

4.1 **Expressões**

As expressões simbólicas em Lisp são conhecidas como *expressões-s*: “uma expressão-s pode ser um átomo ou uma lista.” (LUGER, 2004, p.620).

Em Common Lisp há um processo de leitura das expressões, conhecido como o ciclo *read-eval-print*: leitura de expressões, avaliação de expressões, escrita de resultados. “Expressões são também chamadas de *formas*, especialmente quando são destinadas à avaliação. Uma expressão pode ser simplesmente um símbolo, e neste caso o seu valor é o valor como dado do símbolo.” Meidanis, João. *MC336 - Paradigmas de programação.* (2011, p. 7). PDF. (\*\*\*)

As expressões em Common Lisp são expressões do tipo pré-fixa, aonde o operador vem antes dos argumentos a serem operados. “Talvez seja um estranho jeito de escrever as expressões, mas de fato essa notação é uma das melhores coisas de Lisp”. (GRAHAM, 1996, p. 8, tradução nossa).

Exemplo:

;exemplos de expressoes

>(+ 5 2)

7

>(- 5 (\* 2 5))

-5

>(\* (\* 3 2) (+ 3 2))

30

4.2 **Atribuições**

Em quase todas as linguagens imperativas a atribuição de valores numa variável é algo bem simples. Em Common Lisp as atribuições são feitas através da função *setf* ou *setq*. Estas funções atribuem o valor, ou expressão, digitado a variável sem avaliar o primeiro argumento, que nesse caso é um símbolo (ou variável como conhecemos nas outras linguagens).

Exemplos:

;atribuindo valores as variaveis

>(setf num (+ 2 3)

num 2 (\* 2 3))

6

>num

5

>num2

6

;variavel soma ira receber a soma de duas variaveis

>(setf soma (+ num num2))

11

>num2

11

;variavel n recebe um valor especifico

>(setf n 30)

30

**5. Comandos:**

5.1 **Atribuições**

5.2 **Desvios**

5.3 **Sequenciadores**

5.4 **Chamadas/Retornos**

**6. Abstrações e Parâmetros**

6.1 **Abstrações**

6.2 **Parâmetros**

Em Common Lisp, parâmetros são chamados de listas lambda, escritos em sua “linguagem”, deixando a função mais flexível. Estas listas possuem alguns tipos de parâmetros, tais como:

6.2.1 Número indefinido de parâmetros (*&rest*):

Para escrever uma função que aceite qualquer numero de argumentos, usa-se *&rest* seguido de um parâmetro para segurar a lista de argumentos restantes. Se *&rest* for inserido antes do último parâmetro da lista, a função em uso suportará qualquer quantidade de parâmetros.

Exemplo:

;exemplo de &rest

>(defun foo (x &rest y) y)

FOO

>(foo 10 11 12 13) ;aceita mais de um parametro

(11 12 13)

6.2.2 Parâmetros Opcionais (*&optional*):

“Existe a possibilidade de omitir alguns argumentos, tais parâmetros são chamados de opcionais. Basta colocar *&optional* antes dos parametros que estes se tornarão NIL (vazio).” (GRAHAM, 1996, p. 102, tradução nossa).

Exemplo:

;exemplo de &optional

>(defun filosp (palavra &optional propriedade)

(list palavra ‘e propriedade)) ;propriedades não aparecera

FILOSP

>(filosp ‘vida)

(VIDA E NIL)

6.3 Parâmetros por nome ou palavra-chave (*key*):

Assim como o parâmetro opicional, este parâmetro também faz com que os parâmetros sejam opcionais porém, quando os mesmos forem chamados, não será pela sua posição e sim pela tag simbólica que as procedem. Este parâmetro traz uma vantagem quando se tem vários parâmetros no programa, usar uma palavra-chave ajuda na hora de lembrar do parâmetro.

Exemplo:

> (defun keylist (a &key x y z)

(list a x y z))

KEYLIST

> (keylist 1 :y 2)

(1 NIL 2 NIL)

> (keylist 1 :y 3 :x 2)

(1 2 3 NIL)

**7. Construções de Encapsulamento**

**8. Sistemas de Tipos:**

A presença de herança e polimorfismo permite muitas vezes estender a funcionalidade de um programa sem alterações do código existente. O suporte a orientação a objetos, com herança e polimorfismo, contribui para o reuso de códigos ao longo do programa.

8.1 **Polimorfismo**

O polimorfismo é uma propriedade de Programação Orientada a Objetos, onde uma função pode ter mais de uma forma, ter comportamentos diferentes, dependendo do tipo de seus argumentos. CLOS (COMMON LISP OBJECT SYSTEM) suporta polimorfismo por meio de funções genéricas. (GRAHAM, 2004, p. 677).

Esta propriedade possibilita uma considerável simplificação no processo de programação e de interface com o usuário. É possível escrever um protótipo de uma método, obtendo-se assim uma versão inicial funcional, e modificá-lo ao desenvolver do programa, melhorando sua performance.

8.2 **Sobrecarga**

8.3 **Herança**

A herança é um mecanismo que suporta as abstrações das classes de uma linguagem de programação, permitindo-nos definir classes gerais para a especificação da estrutura de suas especilizações. A CLOS fornece um algoritmo de herança mais robusto. (GRAHAM, 2004, p. 677).