Universidade Positivo
Engenharia da Computação — 11/02/2019 - 12:35:12.4
Inteligência Artificial — Prof Dr P Kantek (pkantek@up.edu.br)
Alocação em LISP — VIVO293a V: 3.25
1
1- _____ / _____ / _____

## Alocação de memória pelo interpretador LISP

*Se você quer ir para Paris, precisa aprender francês. Se quer ir para a Inteligência Artificial, precisa aprender LISP*

O LISP nasce como filho da comunidade de Inteligência Artificial. Na origem, 4 grupos de pessoas se inseriram na comunidade de IA: os lingüistas (na busca de um tradutor universal), os psicólogos (na tentativa de entender e estudar processos cerebrais humanos), matemáticos (a mecanização de aspectos da matemática, por exemplo a demonstração de teoremas) e os informáticos (que buscavam expandir os limites da ciência). O marco inicial é o encontro no Darthmout College em 1956, que trouxe duas conseqüências importantes: a atribuição do nome (IA) e a possibilidade dos diferentes pesquisadores se conhecerem e terem suas pesquisas beneficiadas por uma interfecundação intelectual. A primeira constatação foi a de que linguagens existentes privilegiavam dados numéricos na forma de arrays. Assim, a busca foi a criação de ambientes que processassem símbolos na forma de listas. A primeira proposta foi IPL (Information Processing Language I, por Newel e Simon em 1956). Era um assemblão com suporte a listas. A IBM engajou-se no esforço, mas tendo gasto muito no projeto FORTRAN decidiu agregar-lhe capacidade de listas, ao invés de criar nova linguagem. Nasceu a FLPL (Fortran List Processing Language). Em 1958, McCarthy começou a pesquisar requisitos para uma linguagem simbólica. Foram eles: recursão, expressões condicionais, alocação dinâmica de listas, desalocação automática. O FORTRAN não tinha a menor possibilidade de atendê-las e assim McCarthy junto com Marvin Minsky desenvolveram o LISP. Buscava-se uma função Lisp Universal (como uma máquina de Turing Universal). A primeira versão (chamada LISP pura) era completamente funcional. Mais tarde, LISP foi sofrendo modificações e melhoramentos visando eliminar ineficiências e dificuldades de uso. Acabou por se tornar uma linguagem 100% adequada a qualquer tipo de resolução de problema. Por exemplo, o editor EMACS que é um padrão no mundo UNIX está feito em LISP. Inúmeros dialetos LISP apareceram, cada um queria apresentar a sua versão como sendo a melhor. Por exemplo, FranzLisp, ZetaLisp, LeLisp, MacLisp, Interlisp, Scheme, T, Nil, Xlisp, Autolisp etc. Finalmente, como maneira de facilitar a comunicação entre todos estes (e outros) ambientes, trabalhou-se na criação de um padrão que foi denominado Common Lisp. Como resultado o Common Lisp ficou grande e um tanto quanto heterogêneo, mas ele herda as melhores práticas de cada um de seus antecessores.

**Listas** Uma lista é uma composição de dados. Neste sentido um dado composto (lista) é o antônimo de átomo. Uma lista é um objeto fundamental em LISP. Tudo em LISP pode ser representado como uma lista. Inclusive funções são representadas como listas (são listas!) em LISP.

Listas são representadas de duas maneiras, a externa e a interna. Para o usuário de LISP apenas a representação externa interessa, mas para quem quer conhecer LISP e programá-lo, ambas são importantes.

A representação externa de uma lista começa com um "abre parênteses", segue por uma lista de átomos ou de listas separados por pelo menos um espaço e terminado por um fecha parênteses.

Neste exercício em particular sobre as possíveis maneiras de alocar memória, o que nos interessa mesmo é a representação interna.

Para entender a representação interna, precisa-se conhecer as funções fundamentais de LISP, que são: (CAR $l_x$), e que responde com o primeiro elemento de $l_x$; (CDR $l_y$) que responde com a lista que sobra em $l_y$, após a retirada de seu CAR; (CONS a b) que constrói listas, como em `(cons a nil)` → `(a)` e (eval $l_z$) executa a expressão $l_z$. A partir delas é que a linguagem é desenvolvida.

Seja exemplos de funções lisp:

```
> (defun listasiguais (L1 L2)   ; pergunta se duas listas são iguais
    (cond
       ((null L1) (null L2))
       ((null L2) nil)
       ((not (eql (car L1) (car L2))) nil)
       (t (listasiguais (cdr L1) (cdr L2)))))
LISTASIGUAIS
> (defun remove2 (ato lis)
    (cond
      ((null lis) nil )
      ((eql ato (car lis)) (remove2 ato (cdr lis)))
      (T (cons (car lis) (remove2 ato (cdr lis))))))
REMOVE2
> (defun contaels (lista)
   (cond
    ((null lista) 0)
    ((atom lista) 1)
    (t (+ (contaels (car lista)) (contaels (cdr lista))))
   )
  )
CONTAELS
```

**Representação interna** Os elementos de LISP são representados internamento por uma tabela de símbolos e por um "saco"de células CONS. As células CONS se unem com o auxílio da tabela de símbolos para formar as listas, que são o elemento fundamental em LISP.

Uma célula CONS é um conjunto de 2 endereços que apontam para a tabela de símbolos. Elementos da tabela, por sua vez apontam para celulas CONS que são encadeadas.

O primeiro endereço da célula CONS é chamado CAR, o segundo é chamado CDR. COMMON LISP possui os sinônimos "first"e "rest"mas parece que ninguém os usa.

**Exercício 1** Acompanhe no quadro negro a representação da lista: LT=(TENHO 22 REAIS) Se imaginarmos que a tabela de símbolos começa no endereço 1000 (tamanho de cada entrada=12) e o "saco"de CONS começa em 2000, (tamanho de cada CONS=8), poderiamos ter a seguinte situação:

```
2000=LT                    2000=1012,2008
1012=TENHO                 2008=1024,2016
1024=22                    2016=1036,1048
1036=REAIS
1048=nil
```

**Exercício 2** Acompanhe no quadro negro o desenho e os endereços da lista L=(A (B C) D)

**Exercício 3** Seja o seguinte o conteúdo de memória: (Obs: No endereço 1000 está a tabela de símbolos (tamanho=12) e no endereço 2000 está o "saco"de CONS (tamanho=8)

```
ENDE CONT      ENDE  CAR  CDR        ENDE  CAR  CDR
1000-0580      2000- 2008 2016       2088- 2096 2152
1012-0639      2008- 1060 1180       2096- 2104 2112
1024-1057      2016- 1048 2024       2104- 1144 1180
1036-3358      2024- 2032 2040       2112- 2120 2128
1048-3777      2032- 1168 1180       2120- 1012 1180
1060-4155      2040- 1024 2048       2128- 1036 2136
1072-4450      2048- 1156 2056       2136- 1084 2144
1084-4466      2056- 2064 2088       2144- 1000 1180
1096-JTYA      2064- 2072 2080       2152- 2160 1180
1108-KJFC      2072- 1108 1180       2160- 1096 2168
2000-L1        2080- 1072 1180       2168- 1132 1180
1132-OYGZ
1144-QBDF
1156-RZZH          Este exercício terá como resposta:
1168-UVKJ          ((4155) 3777 (UVKJ) 1057 RZZH ((KJFC) 4450)
1180-nil           ((QBDF) (0639) 3358 4466 0580) (JTYA OYGZ))
1192-t
```

```
Antes de continuar, vamos treinar achar o último elemento de uma lista
(a b c d) => d
((a b) (c d) (e f)) => (e f)
(a (b (c))) => (b (c))
(((a))) => ((a))
```

### ☞ Para você fazer

Construa a lista L1, a partir dos dados abaixo:

| tabela de símbolos | |
|---|---|
| Endereço | Conteúdo |
| 1000 | 1046 |
| 1012 | 1164 |
| 1024 | 1403 |
| 1036 | 2458 |
| 1048 | 3108 |
| 1060 | 4326 |
| 1072 | 4555 |
| 1084 | 4672 |
| 1096 | EKYR |
| 1108 | GCYX |
| 1120 | GZUL |
| 2000 | L1 |
| 1144 | QZHR |
| 1156 | WBAR |
| 1168 | YLCR |
| 1180 | nil |
| 1192 | t |

| saco de CONS | | |
|---|---|---|
| Endereço | CAR | CDR |
| 2000 | 2008 | 2040 |
| 2008 | 2016 | 2024 |
| 2016 | 1120 | 1180 |
| 2024 | 1024 | 2032 |
| 2032 | 1144 | 1180 |
| 2040 | 2048 | 2056 |
| 2048 | 1096 | 1180 |
| 2056 | 1156 | 2064 |
| 2064 | 1012 | 2072 |
| 2072 | 1048 | 2080 |
| 2080 | 2088 | 2096 |
| 2088 | 1084 | 1180 |
| 2096 | 2104 | 2112 |
| 2104 | 1168 | 1180 |
| 2112 | 1000 | 2120 |
| 2120 | 2128 | 2136 |
| 2128 | 1060 | 1180 |
| 2136 | 1036 | 2144 |
| 2144 | 1108 | 2152 |
| 2152 | 1072 | 1180 |

Depois de ter achado a representação externa (parentizada) da lista L1 acima fornecida, informe: O número de elementos de L1 é:_____.

O último elemento de L1 é:
._____.


- 1 - /