PARADIGMA FUNCIONAL: LISP

- Desenvolvida por John McCarthy (1960) para facilitar o trabalho com listas
- Programas em Lisp são as próprias listas
- Tipos de objetos

  Átomos
  - Representados por *strings* de caracteres
  - Se representados por caracteres numéricos são chamados de átomos numéricos (admitem aplicação de funções numéricas)

  Listas
  - Representadas por uma seqüência de átomos e listas separadas por espaço e entre parênteses
  - Exemplos: (a b c) (w (x y z) ( )) (plus 14 12)
  - Lista vazia (*nil*) é considerada um átomo e uma lista

- Modelo de implementação

  Cell model

  Lista é representada através de uma lista de células

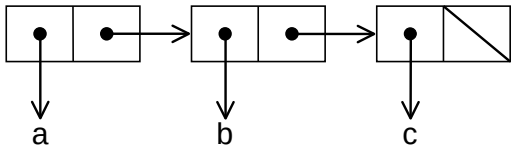
  Componente de cada célula é um ponteiro para o átomo ou lista correspondente

  Ponteiro aponta para o próximo elemento da lista

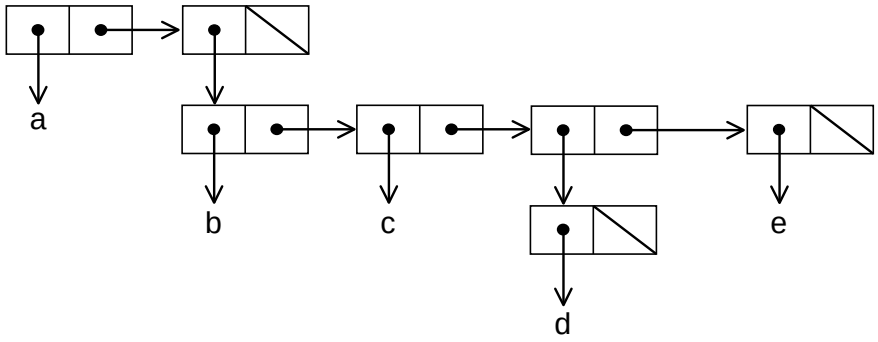
  Terminologia especial para os dois componentes de cada célula
  - *CAR*: parte do dados (ponteiro para o primeiro elemento da lista)
  - *CDR*: parte do ponteiro (ponteiro para as células que representam o resto da lista)

Exemplos:

(a b c)



(a (b c (d) e))



- Funções

  Representadas por listas

  Notação especial: *S-expression*

(<nome_função> <primeiro_parâmetro>....<último_parâmetro>)

Aplicação de uma função é representada através de uma lista

- Primeiro elemento: átomo com o nome da função
- Demais elementos: átomos ou listas representando os parâmetros da função
- Exemplos:
  6 + 9 → (plus 6 9)
  (3+4) + (6+7) → (plus (plus 3 4) (plus 6 7))

- *Built-in Functions*

  Interpretador Lisp considera que cada S-expression é uma expressão para ser avaliada

  Função ***quote*** (ou **‘**) faz com que a expressão seja considerada como uma lista de dados

  Exemplo: ‘(plus 2 3) é uma lista de três elementos

  Função ***car*** retorna o primeiro elemento da lista

Exemplo: (car ‘(a b c)) → a

Função ***cdr*** retorna o elemento para o qual o componente cdr aponta

Exemplo: (cdr ‘(a b c)) → (b c)

Função ***cons*** é usada para construir uma lista

Exemplo: (cons ‘a ‘(b c)) → (a b c)

Função ***atom*** retorna *t* (*true*) se o parâmetro for um átomo e *nil* se não for

Exemplos: (atom ‘a) → t e (atom ‘(a))→nil

Função ***equal*** testa se dois parâmetros são iguais

Função ***cond*** avalia a cláusula de teste até achar uma que não seja *nil*

Função ***setq*** é equivalente a um comando de atribuição

- Funções definidas pelo usuário

(defun <nome_função> (<parâmetro 1>...<parâmetro n>)
<corpo_função>
)

(lambda (<parâmetro 1>...<parâmetro n>)
<corpo_função>
)

- Exemplo: implementação de uma pilha

Pilha é representada como uma lista

Primeiro elemento eqüivale ao topo

Funções fundamentais:

- *empty*: retorna verdadeiro se a pilha está vazia
- *push*: retorna uma lista com um elemento adicionado no topo
- *pop*: retorna uma lista do elemento removido e uma lista da pilha com o elemento removido do topo

```
(defun empty(stack)
            (null stack)
)

(defun push  (stack element)
            (cons element stack)
)

(defun pop    (stack)
            (cond ((empty stack) nil)
                  (t(list (car stack) (cdr stack)))
            )
)
```

PARADIGMA FUNCIONAL: *PATTERN MATCHING*

- Correspondência (ou equivalência) de padrões
- Conceito comum nas LPs Funcionais
- Pode ser explorado nas definições de funções
- Construção de alto nível para execução de condições baseadas em diferentes ações

- Exemplo

```
datatype day = Mon | Tue | Wed | Thu | Fri | Sat | Sun
fun  day_off (Sun) = true
    |day_off (Sat) = true
    |day_off ( _ ) = false
```

Função "day_off" é definida para um certo número de casos (ou situações)

Dependendo do parâmetro na chamada da função o caso apropriado é selecionado

A verificação é feita seqüencialmente a partir do primeiro caso

Neste exemplo a terceira definição corresponde a um argumento qualquer

- Outro exemplo:

```
datatype shape = point
               |circle of real          (* raio *)
               |box of (real * real)    (* largura, altura *)
fun  area (point) = 0.0
    |area (circle r) = pi * r * r
    |area (box(w,h)) = w * h
```

Tipo "*shape*" é uma união disjunta: Unit + Real + (RealxReal) com *tags* "*point*", "*circle*" e "*box*"

Função "*area*"

- Recebe como parâmetro o tipo "*shape*"
- Retorna um número real
- É definida por três equações
- A primeira equação é aplicada somente se o argumento é "*point()*" e portanto corresponde ao padrão "*point*"; neste caso a função retorna 0.0
- A segunda equação é aplicada somente se o argumento é um valor tal como "*circle 5.0*" e então corresponde ao padrão "*circle r*"; neste exemplo "*5.0*" é amarrado a "*r*"
- A terceira equação é aplicada somente se o argumento é um valor tal como "*box(2.0,3.0)*" e então corresponde ao padrão "*box(w,h)*"

- Função pode ser definida através de várias equações
  - Comum por ser uma definição concisa, clara e semelhante à notação matemática
  - Cada equação tem um "lado esquerdo" diferente: contém um *pattern* na posição do parâmetro formal que deve corresponder ao parâmetro real na chamada da função para a equação ser aplicada
- Efeitos
  - Escolhe a ação de acordo com o argumento
  - Amarra nomes do *pattern* com os valores correspondentes (parâmetros) que são utilizados na expressão que define o valor que a função retorna
- *Pattern-Matching*
  - Generalização da passagem de parâmetros convencional
  - Valor do parâmetro real usado para correspondência com o *pattern* do parâmetro formal (o caso selecionado varia a cada chamada)