

# Paradigmas de Programação

Prof. Maicon R. Zatelli

LISP - Programação Funcional  
Estruturas

Universidade Federal de Santa Catarina

Florianópolis - Brasil

2018/1

# LISP

A função **defstruct** permite definir uma estrutura em LISP

```
(defstruct pessoa  
  nome  
  idade  
  cidade  
)
```

- Uma pessoa é composta por nome, idade e cidade

# LISP

Instanciando uma pessoa

```
(setq p1
  (make-pessoa
    :nome "Ana"
    :idade 21
    :cidade "Floripa"
  )
)
```

# LISP

A função **setf** (*set field*) é usada para alterar o valor de campos de uma estrutura.

```
(setf (pessoa-idade p1) 55)  
(setf (pessoa-nome p1) "Fred")
```

- A linha 1 altera a idade da pessoa p1 para 55
- A linha 2 altera o nome da pessoa p1 para Fred

## Árvore binária

```
(defstruct no  
  n  
  esq  
  dir  
)
```

- Um nó possui um valor (n), um nó a esquerda e outro nó a direita.

## Árvore binária

```
(setq minhaArvore
  (make-no
    :n 52
    :esq (make-no :n 32 ;pode omitir o NIL
      :esq (make-no :n 12 :esq NIL :dir NIL)
      :dir (make-no :n 35 :esq NIL :dir NIL)
    )
    :dir (make-no :n 56
      :esq (make-no :n 55 :esq NIL :dir NIL)
      :dir (make-no :n 64 :esq NIL :dir NIL)
    )
  )
)
```

- No código acima é criada uma árvore binária.

# LISP

```
(defun soma (arv)
  (if (null arv)
      0
      (+
        (no-n arv)
        (soma (no-esq arv))
        (soma (no-dir arv))
      )
  )
)
```

- No código acima é feita a soma de todos os elementos de uma árvore.
- Se o nó for NIL, retorna-se 0, caso contrário soma-se o valor daquele nó e também o valor das somas das árvores à esquerda e à direita daquele nó.

# LISP

```
(defun buscaElemento (arv x)
  (if (null arv)
      NIL
      (or
        (= (no-n arv) x)
        (buscaElemento (no-esq arv) x)
        (buscaElemento (no-dir arv) x)
      )
  )
)
```

- No código acima é feita a busca de um elemento na árvore.
- Se o nó for nulo, retorna-se NIL, caso contrário faz-se a função **or** entre o teste de igualdade entre o valor do nó atual e o elemento x e a busca pelo elemento x nas árvores à esquerda e à direita do nó atual.



# LISP

```
(defun minimo (x y) (if (< x y) x y))

(setq INF 1000)

(defun minimoElemento (arv)
  (if (null arv)
      INF
      (minimo
       (no-n arv)
       (minimo
        (minimoElemento (no-esq arv))
        (minimoElemento (no-dir arv)))))))
```

- No código acima é retornado o menor elemento que é presente na árvore.
- Se o nó for nulo, retorna-se INF (infinito), caso contrário retorna o mínimo entre o valor do nó atual e o mínimo entre o mínimo das árvores à esquerda e à direita do nó atual.

# LISP

```
(defun incrementa (arv x)
  (if (not (null arv))
      (progn
        (setf (no-n arv) (+ (no-n arv) x))
        (incrementa (no-esq arv) x)
        (incrementa (no-dir arv) x)
      )
    )
)
```

- No código acima é incrementado um valor  $x$  a todos os valores nos nós da árvore.
- **progn** é uma função que recebe  $n$  funções como argumento e as executa em sequência.

# LISP

```
(defun main()
  (write-line (write-to-string (soma minhaArvore)))
  (write-line (write-to-string (buscaElemento minhaArvore 35)))
  (write-line (write-to-string (buscaElemento minhaArvore 36)))
  (write-line (write-to-string (minimoElemento minhaArvore)))
  (write-line (write-to-string (minimoElemento minhaArvore)))
  (write-line (write-to-string (incrementa minhaArvore 2)))
  (write-line (write-to-string minhaArvore))
)

(main)
```

# LISP - Alguns Links Úteis

- <https://www.tutorialspoint.com/lisp/index.htm>
- [https://www.tutorialspoint.com/lisp/lisp\\_structures.htm](https://www.tutorialspoint.com/lisp/lisp_structures.htm)

Ver atividade no Moodle