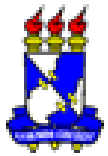
# Framework para uma Família de LP

Prof. Alberto Costa Neto
alberto@ufs.br

Linguagens de Programação



**Departamento de Computação**
**Universidade Federal de Sergipe**

# Linguagens

- Linguagem de Expressões 1 (LE1)
- Linguagem de Expressões 2 (LE2)
- Linguagem Funcional 1 (LF1)
- Linguagem Funcional 2 (LF2)
- Linguagem Funcional 3 (LF3)
- Linguagem Imperativa 1 (LI1)
- Linguagem Imperativa 2 (LI2)
- Linguagem Orientada a Objetos 1 (LO1)

# LF1

## Linguagem Funcional 1

(Funções de primeira ordem)

# LF1 - Características

- Estende LE2 com **funções parametrizadas e recursivas** (de **primeira ordem**, funções não são valores)
- O **corpo** de uma função é uma **expressão**
- Um **programa continua sendo uma expressão**, que pode envolver invocação (aplicação) de função
- A **aplicação** da função a um argumento **retorna um valor**
  - Aplicação também é uma expressão

# LF1 – Características (cont.)

- Introduz a expressão condicional:
  **if cond then e1 else e2**
  - Caso a **cond** seja true, **e1** é avaliada e retornada. Caso contrário isto é feito com **e2**
- Eager evaluation (parâmetros)
- Utiliza escopo dinâmico

# Ambientes (Contextos)

- O ambiente de execução inclui dois componentes:
  - mapeamento de identificadores em valores (LE2)
  - mapeamento de identificadores (nomes de função) em definições de função
- O ambiente de compilação possui apenas um mapeamento de identificadores em tipo
  - O conceito de tipo é estendido para funções
    - Tipo da função é representado pelo produto cartesiano dos tipos dos parâmetros e o tipo do resultado produzido pela função

# LF1 - Sintaxe

Programa ::= Expressao

Expressao ::= Valor | ExpUnaria | ExpBinaria | ExpDeclaracao | Id
    | Aplicacao | IfThenElse

ExpDeclaracao ::= "let" DeclaracaoFuncional "in" Expressao

DeclaracaoFuncional ::= DecVariavel | DecFuncao
    | DeclaracaoFuncional "," DeclaracaoFuncional

DecFuncao ::= "fun" ListId "=" Expressao

Aplicacao:= Id "(" ListExp ")"

IfThenElse ::= "if" Expressao "then" Expressao "else" Expressao

DecVariavel ::= ...

Valor ::= ...

ExpUnaria ::= ...

ExpBinaria ::= ...

# LF1 - Exemplos

```
let fun ident x = x in ident(11)

let fun soma x y = x + y in soma(2,4)

let fun prod x y =
    if y == 0 then 0 else x + prod(x, y-1)
       in prod(3,5)
```

# LF1 – Exemplos (cont.)

```
let fun prod x y =
   if y == 0 then 0 else x + prod(x, y-1)
    in let var k = 5,
            fun fat n =
               if n == 0 then 1 else prod(n, fat(n-1))
               in fat(k)
```

# Exercícios

1. Modifique a implementação da Linguagem Funcional 1 de forma que o escopo das variáveis seja estático.
2. Atualmente, a verificação de tipos é implementada através de um algoritmo de inferência, já que as variáveis são declaradas sem os tipos explícitos; inclua tipos explícitos nas declarações de variáveis e funções e reimplemente a verificação de tipos.
   - Possível sintaxe:

     let var z : Int = 2, fun f : Int -> Int . f x = x + 1 in …

# LF2

## Linguagem Funcional 2

(Funções de Alta Ordem)

# LF2 - Características

- LF2 estende LF1 introduzindo **funções de alta ordem**
- Funções passam a ter **status de valor**
  - Podem ser passadas como parâmetro, retornadas de funções, …
- Requer uma extensão do conceito de valor de LF1
  - Criado um tipo de valor abstrato (valor função)
- A aplicação de uma função **deixa de ser um identificador** e **passa a ser uma expressão** (gera a função a ser aplicada)

# LF2 - Sintaxe

Programa ::= Expressao

Expressao ::= ...

Valor ::= ValorConcreto | ValorAbstrato

ValorAbstrato ::= ValorFuncao

ValorFuncao ::= "fn" Id Id "." Expressao

ValorConcreto ::= ...

Aplicacao := Expressao "(" ListExp ")"

# LF2 - Exemplos

```
let fun id x = x in id
```

=> Retorna a função identidade

```
let fun suc x = x + 1
    in let fun id x = x
        in id(suc)
```

=> Retorna a função sucessor

# LF2 – Exemplos (cont.)

Composição de funções

```
let fun pred x = x -1, fun suc x = x + 1
   in let fun comp f g x = f(g(x))
      in comp(pred,suc,1)
```

=> Retorna 1

```
let fun pred x = x -1, fun suc x = x + 1
   in let fun comp f g = fn x . f(g(x))
      in comp(pred,suc)
```

=> Retorna uma função (usa função anônima / expressão lambda)

# LF3

## Linguagem Funcional 3

(Listas)

# LF3 - Características

- Introduz o conceito de listas
- Operações sobre listas
  - Head
  - Tail
  - Cons (:)
  - Concatenação (^^)
- Compreensão de Listas

# LF3 – Sintaxe

ValorConcreto ::= ValorInteiro | ValorBooleano
| ValorString | ValorLista
ExpUnaria ::= "-" Expressao | "not" Expressao
| "length" Expressao | head(Expressao) | tail(Expressao)
| ExpCompreensaoLista

ExpCompreensaoLista ::= "[" Expressao Gerador "]"
| "[" Expressao Gerador Filtro "]"
Gerador ::= “for” Id “in” Expressao
| “for” Id “in” Expressao “,” Gerador
Filtro ::= “if” Expressao

# LF3 – Sintaxe (cont.)

ExpBinaria ::= Expressao "+" Expressao
| Expressao "-" Expressao | Expressao "*" Expressao
| Expressao ">" Expressao | Expressao "<" Expressao
| Expressao "and" Expressao | Expressao "or" Expressao
| Expressao "==" Expressao | Expressao "++" Expressao
| Expressao ".." Expressao | Expressao ":" Expressao
| Expressao "^^" Expressao

# LF3 - Exemplos

1:2:3:[] => Retorna [1, 2, 3]

head([1, 2, 3]) => Retorna 1

tail([1, 2, 3]) => Retorna [2, 3]

[1, 2, 3] ^^ [4] => Retorna [1, 2, 3, 4]

[x*2 for x in 1..3] => Retorna [2, 4, 6]

[x*2 for x in 1..10 if x < 5] => Retorna [2, 4, 6, 8]