Paradigmas de Linguagens de Programação

Linguagens de Programação Funcionais

Cristiano Lehrer, M.Sc.

ybadoo

Introdução (1/3)

- É uma categoria de linguagens não-imperativas.
- Imperativas:
  - Uso eficiente das arquiteturas de computadores de Von Neumann.
- A arquitetura não deveria ser uma restrição no processo de desenvolvimento de software.
- Alguns outros paradigmas existem:
  - Não são eficientes.
  - Não dominam o mercado.

Introdução (2/3)

- O paradigma de programação funcional é baseado em funções matemáticas.
- LISP iniciou como uma linguagem funcional pura, mas adquiriu algumas características imperativas.
- Outras linguagens:
  - Schema, Common LISP, ML, Miranda.

Introdução (3/3)

- Uma linguagem funcional provê:
  - Conjunto de primitivas.
  - Conjunto de formas funcionais para construir funções complexas a partir das funções primitivas.
  - Operações de aplicação de funções.
  - Alguma forma de armazenamento de dados.

LISP

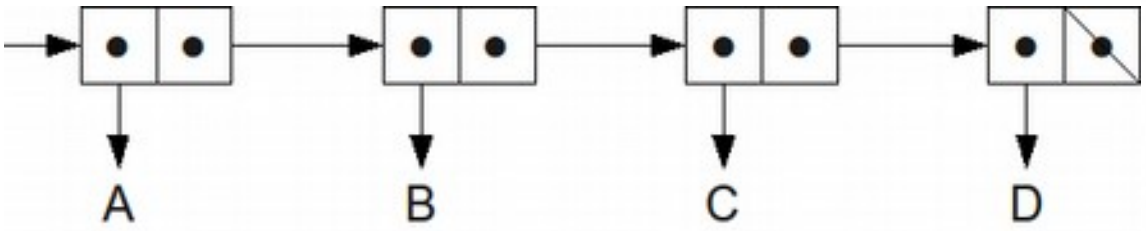
- Família de linguagens de programação concebida por John McCarthy em 1958.
- A mais antiga e mais amplamente usada.
- LISP – **Lis**t **P**rocessing (a lista é a estrutura de dados fundamental desta linguagem).
- Com exceção da primeira versão, todos os dialetos incluem algumas características de linguagens imperativas:
  - Variáveis.
  - Instruções de atribuição.
  - Iteração.

Tipos de Dados e Estruturas (1/3)

- LISP é uma linguagem sem tipos.
- Dois tipos de objetos de dados:
  - Átomos:
    - São os símbolos de LISP.
  - Listas:
    - São as estruturas de dados.

Tipos de Dados e Estruturas (2/3)

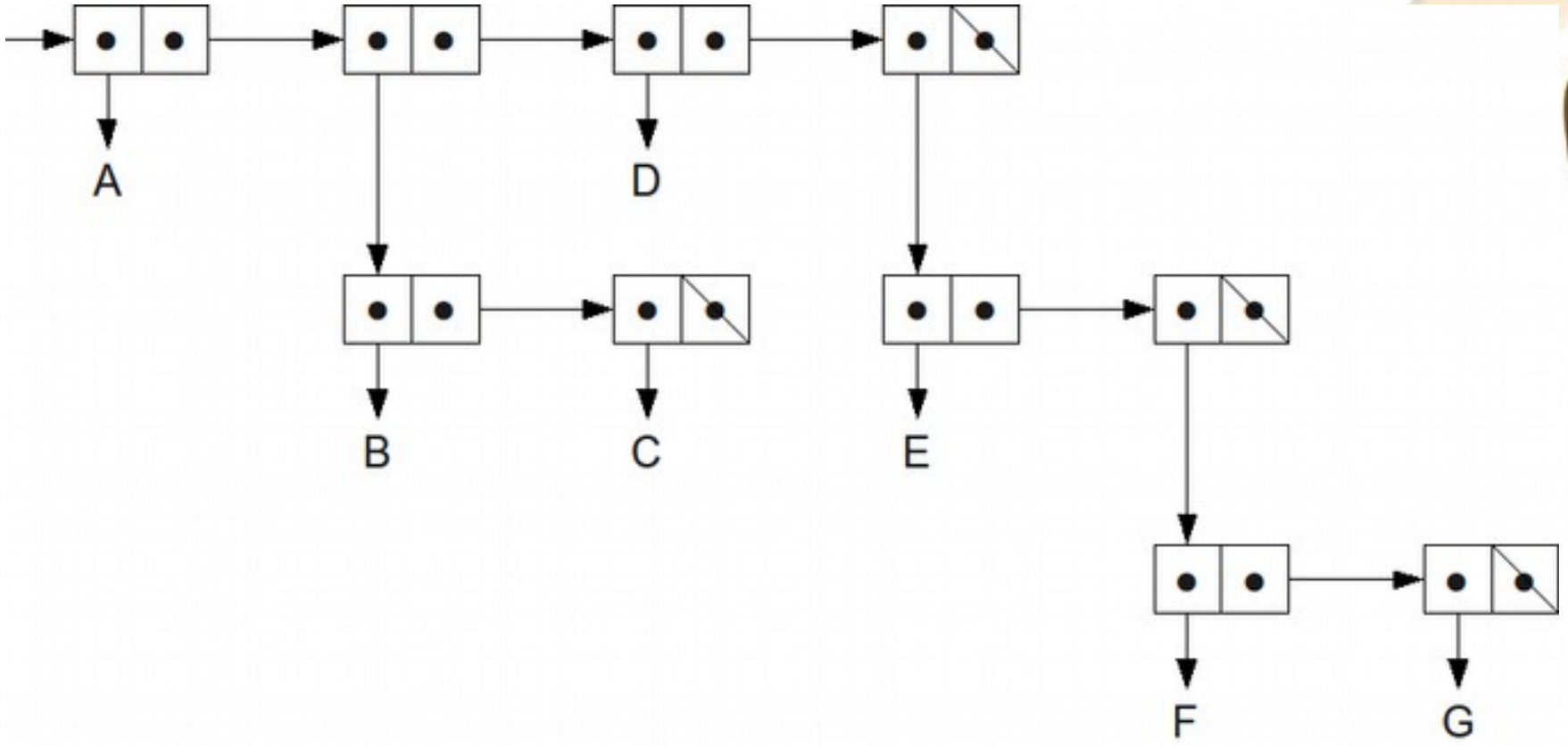
- Listas são especificadas delimitando seus elementos por parêntesis:
  - (A B C D)



ybadoo

Tipos de Dados e Estruturas (3/3)

- Estruturas de listas aninhadas são também permitidas:
  - (A (B C) D (E (F G)))



ybadoo

Aplicações

- LISP é versátil e poderosa.
- LISP foi desenvolvida para computação simbólica e aplicações de processamento de listas.
- Editor de texto escrito em LISP: EMACS
- Cálculos simbólicos: MACSYMA
- Ensino introdutório de programação.
- LISP na Inteligência Artificial:
  - Sistemas Especialistas.
  - Representação do conhecimento.
  - Aprendizado de máquina.
  - Processamento de linguagem natural.
  - Sistemas de treinamento inteligente.
  - Modelagem da fala e visão.

Comparação

- Linguagem Imperativa:
  - Gerência de variáveis e atribuição de valores.
  - Eficiência.
  - Construção de programas requer mais tempo e esforço.
- Linguagem Funcional:
  - Não se preocupa com variáveis.
  - Ineficiência.
  - Programação em altíssimo nível:
    - Não requer muito tempo e esforço (principal vantagem).
  - Sintaxe muito simples:
    - Estrutura de listas.
  - Semântica também simples:
    - Tudo são listas.

Exemplo Introdutório de LISP

```
> 14 ;calcule o valor deste símbolo
 14
> '14 ;dê o símbolo sem calcular o valor
 14
> (setq a 20) ;dê o valor 20 a variável 'a'
 20
> a ;valor de a
 20
> 'a ;volte o símbolo sem calcular o valor
 A
> (+ 1 2 3 4 5) ;dê a soma destes números
 15
> (+ 1(- 12 6)) ;calcule
 7
> (defun somasub (a b c)(+ a (- b c))) ;define uma nova função
 SOMASUB ;responde que entendeu a nova função
> (somasub 1 12 6)
 7
> (exit) ;sai do LISP
```

Função EVAL

- Se nada for dito, LISP supõe que todo átomo é o nome de uma função e para indicar que não é o caso, existe a função inversa do **eval**, chamado **quote**, representada frequentemente por uma aspa simples:

```
> (a b c)
 Error: The function "A" is undefined.
> '(a b c)
 (A B C)
> (quote(a b c))
 (A B C)
```

Seletores CAR e CDR

- `(car lista)`
  - Retorna o primeiro elemento de uma lista.
- `(cdr lista)`
  - Retorna a lista sem o primeiro elemento.

```
> (car '(a b c))
 A
> (car '((a b) c d))
 (A B)
> (car 'a)
 Error: The argument to CAR is not a CONS
> (cdr '(a b c))
 (B C)
> (cdr '())
 NIL
> (car (cdr '(meu gato preto está dormindo)))
 GATO
```

Atribuição

- Corresponde ao que em outras linguagens é uma inicialização de variável, bom costume em muitas linguagens, tais como Pascal, C++ e Java, e mau estilo em LISP.

```
> (set 'a '(s d e))
 (S D E)
> (car a)
 S
> (car 'a)
 Error: The argument to CAR is not a CONS
```

Função Construtora

- Esta função tem por argumentos um átomo ou lista e coloca o átomo ou lista como primeiro elemento da nova lista.

```
> (cons 'meu '(gato preto))
  (MEU GATO PRETO)
> (cons '(a) '(b))
  ((A) B)
> (cons '(gato preto) '(sobre PC))
  ((GATO PRETO) SOBRE PC)
> (append '(a s) '(d e f))
  (A S D E F)
```

Predicados (1/3)

- **atom**
  - Retorna **T** se o que se segue é um átomo, e **NIL** caso contrário.

```
> (set 'a '(s d e))
 (S D E)
> (atom a)
 NIL
> (atom 'a)
 T
> (atom ())
 T
> (atom '())
 T
> (atom '(()))
 NIL
```

Predicados (2/3)

- **null**
  - Retorna **T** se o que se segue é **NIL**, e **NIL** caso contrário.

```
> (set 'a '(s d e))
 (S D E)
> (null a)
 NIL
> (null 'a)
 NIL
> (null '())
 T
> (null (cdr '(b)))
 T
```

Predicados (3/3)

- **eql**
  - Determina se dois elementos atômicos são iguais;
  - Dado listas por argumento retornará **NIL**, mesmo que as listas sejam iguais.

```
> (set 'a '(s d e))
 (S D E)
> (eql (car a) 's)
 T
> (eql '(a s d) '(a s d))
 NIL
> (eql 'ram 'rom)
 NIL
> (eql a '(s d e))
 NIL
```

Condicional (1/2)

```
(cond (causa_1 consequencia_1)
      (causa_2 consequencia_2)
      (causa_n consequencia_n))
```

```
> (setq y 70)
 70
> (cond ((> y 65)(setq j 4))
        ((> y 21)(setq j 3))
        ((> y 10)(setq j 2))
        (T       (setq j 1)))
 4
```

Condicional (2/2)

```
(if (test)
    (then part)
    (else part))
```

```
> (setq y 70)
 70
> (if (> y 65)
      (setq j 4)
      (if (> y 21)
          (setq j 3)
          (if (> y 10)
              (setq j 2)
              (setq j 1))))
 4
```

Funções (1/5)

```
(defun nome (arg_1 arg_2 arg_n)
        tarefa_1
        tarefa_2
        tarefa_n)
```

```
> (defun quadrado(x)
    (* x x))
 QUADRADO
> (quadrado 5)
 25
```


ybadoo

Funções (2/5)

```
> (defun fatorial(n)
   (if (zerop n)
       1
       (* n (fatorial(- n 1)))))
 FATORIAL
> (fatorial 3)
 6
> (fatorial 10)
 3628800
```


ybadoo

Funções (3/5)

```
> (defun conta(lst)
    (if (null lst)
        0
        (+ 1 (conta (cdr lst)))))
 CONTA
> (conta '(a b c))
 3
```


ybadoo

Funções (4/5)

```
> (defun soma(lst)
    (if (null lst)
        0
        (+ (car lst) (soma (cdr lst)))))
  SOMA
> (soma '(1 2 3))
 6
```

Funções (5/5)

```
> (defun pertence(elt lst)
   (if (null lst)
       nil
       (if (eql elt (car lst))
       T
       (pertence elt (cdr lst)))))
 PERTENCE
> (setq lista '(a s d f g))
 (A S D F G)
> (pertence 'd lista)
 T
> (pertence 'x lista)
 NIL
```

Executar comandos LISP no JAVA

```
Jatha myLisp = new Jatha(false, false);
myLisp.init();
myLisp.start();
System.out.println(myLisp.eval("(+ 1 2 3 4 5)"));
// 15
```


ybadoo

Inserir Novas Primitivas (1/6)

```
public class Even extends LispPrimitive {
  public Even(Jatha lisp) {
    super(lisp, "EVEN", 1);
  }
  public void Execute(SECDMachine machine) throws CompilerException {
    LispValue arg1 = machine.S.pop();
    if(arg1.basic_numberp()) {
      long number1 = (long)arg1.toJava();
      if((number1 % 2) == 0) {
        machine.S.push(f_lisp.T);
      }
      else {
        machine.S.push(f_lisp.NIL);
      }
    }
    else {
      machine.S.push(f_lisp.NIL);
    }
    machine.C.pop();
  }
}
```


ybadoo

Inserir Novas Primitivas (2/6)

```
Jatha myLisp = new Jatha(false, false);
myLisp.init();
myLisp.start();
myLisp.COMPILER.Register(new Even(myLisp));
System.out.println(myLisp.eval("(even 8)"));
// T
System.out.println(myLisp.eval("(even 9)"));
// NIL
```

Inserir Novas Primitivas (3/6)

```
public class Remainder extends LispPrimitive {
  public Remainder(Jatha lisp) {
    super(lisp, "%", 2);
  }
  public void Execute(SECDMachine machine) throws CompilerException {
    LispValue arg2 = machine.S.pop();
    LispValue arg1 = machine.S.pop();

    long number1 = (long)arg1.toJava();
    long number2 = (long)arg2.toJava();

    machine.S.push(f_lisp.makeBignum(number1 % number2));

    machine.C.pop();
  }
}
```


ybadoo

Inserir Novas Primitivas (4/6)

```
Jatha myLisp = new Jatha(false, false);
myLisp.init();
myLisp.start();
myLisp.COMPILER.Register(new Remainder(myLisp));
System.out.println(myLisp.eval("(% 8 3)"));
// 2
System.out.println(myLisp.eval("(% 8 2)"));
// 0
```


ybadoo

Inserir Novas Primitivas (5/6)

```
public class Sequence extends LispPrimitive {
  public Sequence(Jatha lisp) {
    super(lisp, "SEQUENCE", 1);
  }
  public void Execute(SECDMachine machine) throws CompilerException {
    LispValue arg = machine.S.pop();

    long number = (long)arg.toJava();

    List<LispValue> numbers = new LinkedList<>();

    for(int i = 0; i <= number; i++) {
      numbers.add(f_lisp.makeBignum(i));
    }

    machine.S.push(f_lisp.makeList(numbers));

    machine.C.pop();
  }
}
```


ybadoo

Inserir Novas Primitivas (6/6)

```
Jatha myLisp = new Jatha(false, false);
myLisp.init();
myLisp.start();
myLisp.COMPILER.Register(new Sequence(myLisp));
System.out.println(myLisp.eval("(SEQUENCE 8)"));
// (0 1 2 3 4 5 6 7 8)
```