Programação Funcional

- Alguns paradigmas: imperativo, lógico, orientado a objetos, funcional
- Características centrais da programação funcional:
  - Funções de alta ordem
  - Recursão
  - Abstração lambda
- Outras características de algumas linguagens funcionais:
  - Currying
  - Casamento de padrões
  - Compreensão de listas
  - Avaliação preguiçosa

Imperativo

- Programa = sequência de comandos
- Computação = execução dos comandos

```
/* C */
unsigned fat (unsigned n) {
  unsigned i;
  unsigned f = 1;
  for (i=1; i<n; i++)
    f *= i;
  return f;
}
```

Lógico

- Programa = axiomas
- Computação = prova construtiva de um objetivo

```
% Prolog
% Note o uso de casamento de padrões (fat é declarado duas
% vezes, com dois padrões de argumentos), e o uso de
% recursividade.
fat(0,1).
fat(N,F) :- N1 is N-1, !,
            fac(N1, F1), !,
            F is N * F1.
```

Orientado a Objetos (puro)

- Programa = classes, objetos, mensagens
- Computação = envio de mensagens entre objetos

```
“Smalltalk – primeiro recursivo, depois iterativo, mas
 sempre através do envio de mensagens.”
rfac: n
  (n = 0)
    ifTrue:  [ ^1 ]
    ifFalse: [ ^ n * (self fac: n - 1) ]

ifac: n
  | i |
  i := n.
  v := 1.
  (n > 0)
    whileTrue [ v := v * i ]
  ^v.
```

Funcional

- Programa = funções
- Computação = avaliação das funções
- Baseado no λ-Cálculo

```
;; Scheme:
(define (fac n)
  (cond ((= n 0)
         1)
        (#t
         (* n (fac (- n 1))))))

-- Haskell:
fat 0 = 1
fat x = x * fat (x - 1)
```

Outras idéias relacionadas

- Arrays (APL, J)
- Programação Concorrente (Occam, Erlang)
- Programação orientada a pilha, no estilo das calculadoras HP (Postscript, Joy, FORTH)

Um programa FORTH

```
: FAT recursive          \ definimos a palavra “FAT”
  DUP 1                  \ duplica topo, empilha 1,
    >                    \ compara 2 primeiros
    IF                   \ se maior,
      DUP 1 -            \ dup, empilha 1, subtrai,
      FAC *              \ fat (topo), mult. Pelo anterior
    ELSE
      DROP 1             \ se <= 1, jogue for a e empilhe 1
    ENDIF
;
```

Recursão

- No estilo puramente funcional, é a única forma de implementar repetições

```
;; Common Lisp:
(defun lst-size (lst)
  (if (null lst)
      0
      (1+ (lst-size (cdr lst)))))
```

Funções de alta ordem

- Funções são objetos de primeira classe

```
;; Common Lisp:
(setq a #(1 2 3 4 5 6 7 8))
(sort a #'<)
#(1 2 3 4 5 6 7 8)

(sort a #'>)
#(8 7 6 5 4 3 2 1)
```

Map & Reduce

- Map: aplica função a todos os elementos de sequência
- Reduce: idem, mas acumula valores, gerando um só

```
;; Common Lisp:
(mapcar #'(lambda (x) (* 2 x))
        '(2 4 6))
#(4 8 12)

(reduce #'+ '(2 4 6))
12
```

λ-abstraction

- “λ x . 2x+3” troca “x” por “x+3”
- Usamos para definir funções anônimas

```
;; Em Common Lisp:
(setq a #(1 2 3 4 5 6 7 8))
(sort a #'<)
#(1 2 3 4 5 6 7 8)

(sort a #'>)
#(8 7 6 5 4 3 2 1)

(sort a #'(lambda (x y)
            (and (oddp x)
                 (evenp y))))
#(7 5 3 1 8 6 4 2)

(setq a #'(lambda (x)
              (/ x 10)))
#<FUNCTION :LAMBDA (X) (/ X 10)>
(funcall a 1000)
100
```

Currying

- F: A x B -> C
- F': A -> (B -> C)

```
;; Esta função implementa
;; Currying em Common Lisp:
(defun curry (function
              &rest args)
  (lambda (&rest more-args)
    (apply function
      (append args
              more-args))))
```

```
;; Usando a função anterior:
(defun soma3 (x y z)
  (+ x y z))

(curry #'soma3 10)
         40 50)
#<FUNCTION :LAMBDA
   (&REST MORE-ARGS)
   (APPLY FUNCTION
    (APPEND ARGS MORE-ARGS))>

(setq soma10xy (curry #'soma3
10))

(funcall soma10xy 1 2)
13
```

Casamento de padrões

- Não vem “de fábrica” em Lisp, mas pode ser incluído
- Existente em Prolog, Haskell, ML, Ocaml...

```
-- Haskell
fat 0 = 1
fat x = x * fat (x - 1)
```

List Comprehension

- Em Haskell, podemos usar notação de conjuntos para produzir listas

```
-- Haskell:

[ (x,y) | x <- [1,3,5],
          y <- [2,4,6],
          x<y ]
[ (1,2),(1,4),(1,6),
  (3,4),(3,6),(5,6)]
```

Macros

- Programas que produzem programas
- Como macros de C, mas com inteligência de um programa
- Decisão sobre que código gerar *antes* da execução

```
;; Common Lisp. Exemplo hipotético
;; (não rodaria, porque linuxp não
;; existe).
(defmacro matrixmult (x y)
  (if (linuxp) ; funcao hipotetica
      (fast-linux-mat-mult x y)
      (generic-mat-mult x y)))

Em Linux,

(matrixmult a b)
=> (fast-linux-mat-mult a b)

Em outro sistema,

=>(generic-mat-mult a b)
```

Mais...

- Avaliação preguiçosa (Haskell)
- Sistema de condições (Common Lisp)
- CLOS (sistema de objetos de Lisp)
- Modelo de concorrência de Erlang

Características

	Família Lisp	Haskell	ML/OCaml	Erlang
Sistema de tipos	normalmente dinâmico	estático	estático	estático
Efeitos colaterais	intercalados no programa	isolados (suporta programação funcional pura)	intercalados no programa	intercalados no programa
Avaliação	estrita	preguiçosa	estrita	estrita
Alguns pontos fortes (não todos)	metaprogramação (macros), dinamismo	verificabilidade	velocidade	concorrência

Três dialetos de Lisp

	Lush	Scheme	Common Lisp
	Razoável	Limpo	Sujo, estranho em um primeiro contato
Tamanho da linguagem	Razoável	Minimalista (pelo menos até o R5RS)	Astronômico
Desenvolvimento			Ambiente interativo
Execução		Runtime pequeno	Runtime que inclui compilador, debugger e a pia da cozinha
Bibliotecas	As de C	Boa quantidade, mas nem tanto	Muitas!
Estilos suportados out-of-the-box	Funcional	Funcional	Funcional, imperativo, OO
Escopo de variáveis	Dinâmico	Estático	Dinâmico ou estático
Namespaces	\	\	7 (variáveis, rótulos, funções e outras entidades podem ter o mesm
Pontos fortes	Mescla/interação com C; velocidade	Continuações	Sistema de exceções; ambiente interativo (SLIME); configurabilida depuração, visualização do assembly gerado e muitas outras utilid integradas no ambiente

Alguns recursos

- comp.lang.functional
- lisp.org, comp.lang.lisp, Planet Lisp, cliki.net
- schemers.org, comp.lang.scheme
- lush.sourceforge.net
- haskell.org
- erlang.org

Outras Linguagens Funcionais

- Erlang (programação concorrente e distribuída)
- Lush (Lisp altamente eficiente)
- Arc (Lisp “conciso” de Paul Graham)
- ML, OCaml (outra linguagem funcional)
- Qi (sistema de tipos mais flexível que há)
- Q (linguagem baseada em reescrita de termos)

Bibliografia -- Lisp

título	autor	apelido	comentário
Practical Common Lisp	Peter Seibel	PCL	Excelente tutorial, com foco em aplicações práticas
On Lisp	Paul Graham		Lisp avançado: técnicas para uso de macros
Paradigms of AI Programming with Common Lisp	Peter Norvig	PAIP	Lisp avançado. A parte de IA é defasada, mas as técnicas em Lisp são boas
Object Oriented Programming in Common Lisp	Sonya Keene		Fala apenas do CLOS, o sistema de objetos de Common Lisp, mas diz tudo o que você pode querer saber, exceto sobre o protocolo de metaobjetos
The Art of the Metaobject Protocol	Gregor Kiczales		Tudo sobre o protocolo de metaobjetos
The Scheme Programming Language	Dybvig		Tutorial sobre Scheme
Structure and Interpretation of Computer Programs	Abelson, Sussman	SICP	Vai do zero ao muito avançado usando Scheme. Um dos melhores livros já escritos sobre prática de programação
Lisp in Small Pieces	Christian Queinnec	LiSP	Como escrever sua própria implementação de Lisp

Bibliografia -- Haskell

título	autor	comentário
Yet Another Haskell Tutorial		Um tutorial muito bom, livre
Real World Haskell		Parece ter foco em aplicações reais. Livre, por enquanto
The Haskell Road to ...		Para quem gosta de matemática
Haskell: uma abordagem prática	Sá & Silva	Não tão prática; um tutorial bastante básico
The Haskell School of Expression	Paul Hudak	Tutorial, usando gráficos e música

Mais livros...

título	autor	comentário
An Introduction to Lambda Calculi for Computer Scientists	Chris Hankin	Uma introdução curta e simples ao lambda-cálculo
Purely Functional Data Structures	Chris Okasaki	Estruturas de dados no estilo puramente funcional

Sugestões de projeto para aprender mais

- Para aprender uma das lingaugens, além dos tutoriais recomendados, tente usá-la para implementar algo interessante:
  - Um interpretador para a versão R5RS de Scheme (a R6RS é muito grande)
  - Um *blog engine*
  - Um pequeno editor de textos
  - Utilitários (seu próprio grep/sort/find/etc)
  - Um jogo do tipo MUD
  - Algo mais que você goste!