Lenguajes de programación

Introducción al paradigma de programación Funcional

Paradigma Funcional

- Basado en la evaluación (o aplicación) de funciones.
- Mayor nivel de abstracción al programar.
- Planteamiento de la solución a un problema, basado en un "QUE" y no en "COMO".

Evaluación de funciones

Todo programa es una función, que dadas ciertas entradas, genera ciertos resultados.

Definición de la función f

$f(x, y) = 2x - 5y$

Parámetros

Cuerpo de la función

Aplicación o evaluación de la función f

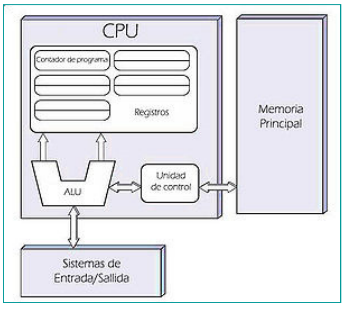
$f(7, 2) \Rightarrow 4$

Argumentos

Resultado

Mayor nivel de abstracción

- Alejado del modelo de la máquina de Von Neumann.
- No construye soluciones mediante modificaciones de estado.
- Resuelve los problemas principalmente mediante la combinación de funciones.
- Modularidad obligada: cada función es un módulo.



Diferencias entre Paradigmas

Estilo Imperativo

```
Function Factorial(n : int): int;
Var  fact, j : int;
Begin
  fact = 1;
  For j = 2 to n do
     fact = fact * j;
  Return (fact);
End;
```

Estilo Funcional

```
Function Factorial(n : int): int;
Begin
  If n == 0 then
     Return (1)
  else
     Return (n * Factorial(n-1));
End;
```

Ventaja Consecuente: TRANSPARENCIA REFERENCIAL

Un módulo con el estilo funcional (o función):

- NO depende de variables o valores globales.
- No utiliza variables locales.
- No realiza asignaciones.
- Controla su ejecución con decisiones y recursividad.

Fácil lectura, mantenimiento, paralelización y comprobación.

Fundamentación teórica: CALCULO LAMBDA

- Propuesto por Alonso Church en 1941.
- Notación especial para definir funciones sin nombre.
- Las funciones pueden ser argumentos de otras funciones, o bien, resultado de otras funciones.

$$\lambda x . \lambda y . x + y$$

Lenguajes Funcionales

- LISP
  - John McCarthy, 1958
    - FranzLisp
    - CommonLisp
- APL (Array programming language)
  - Kenneth Iverson,1964
- LOGO (Lisp without parenteses)
  - Daniel G. Bobrow, Wally Feurzeig & Seymour Papert, 1967
- SCHEME (Dialecto de Lisp)
  - Guy Steels & Gerald Sussman, 1975
- FP (Function Programming)
  - John Backus, 1977
- ML (MetaLanguage)
  - Robin Milner, 1973
- MIRANDA
  - David Turner, 1985

Principales aplicaciones

- Inteligencia Artificial
- Problemas científicos
- Sistemas para comprobación matemática
- Manipulación simbólica
- Lógica
- Programación concurrente
- …

Otras Características de los lenguajes funcionales

- **Funciones como elementos de primer orden.**
  - Se pueden utilizar como argumentos.
  - Se pueden generar como resultado.
  - Pueden ser manipuladas como los datos.
- **Administración de la memoria con bindings dinámicos.**
  - Los datos NO están restringidos en su capacidad.
  - La única estructura de datos que se provee es la **LISTA**.
- **La evaluación de funciones está sujeta a las condiciones de ejecución (lazy evaluation).**
- **Normalmente utilizan un INTERPRETE como traductor.**

Desventajas

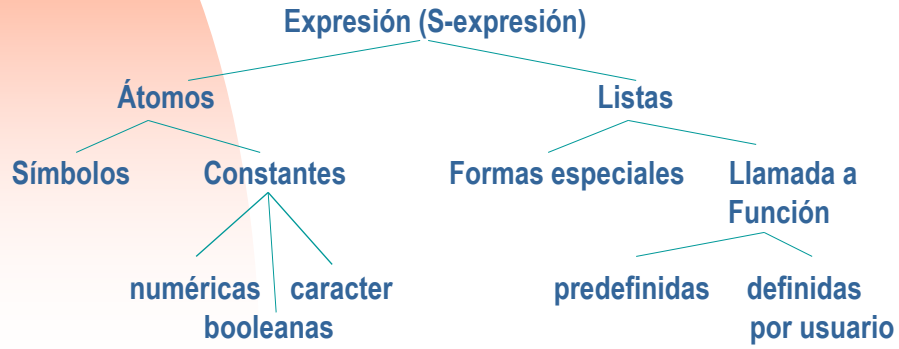
- **Menor eficiencia en la ejecución de programas.**
  - Interpretación
  - Traducción de un nivel mayor.
  - Acceso a estructuras de datos: Listas vs. Arreglos
- **Conflicto con el manejo de operaciones de entrada/salida.**
  - La solución han sido los lenguajes híbridos o no puros.
- **"Lesión cerebral" causada por la forma tradicional (procedural/imperativa) de programar.**

Primer lenguaje para Programación Funcional

- **Racket** (anteriormente llamado PLT Scheme)
  - Lenguaje de programación multi-paradigma de propósito general de la familia Lisp/Scheme.
  - Página web: http://racket-lang.org/

Expresiones en Scheme

- La evaluación de expresiones es la base con la que opera el interpretador de Scheme.



Átomos

- La evaluación de una constante, genera como resultado el valor de la propia constante.
- *Ejemplos: 876, -92, 0.123, -4.508*
- La evaluación de un símbolo, genera como resultado el valor asociado a ese símbolo.
- Un símbolo es un identificador que se construye con cualquier caracter, excepto: ( ) [ ] { } ; , ‘ “ # \

Listas

- Elemento básico de trabajo en el lenguaje.
- Sintaxis:

  <Lista> ::= (<elem>)

  <elem> ::= atomo <elem>

  <elem> ::= <Lista> <elem>

  <elem> ::= ε

Listas como llamada a funciones (procedimientos)

- El primer elemento en la lista corresponde al nombre de la función que se llama.
- Los siguientes elementos corresponden a los argumentos de la llamada.
- El intérprete usa un ***orden aplicativo*** en la evaluación:
  Primero se evalúan los argumentos, y después se aplica la función.

Funciones (procedimientos) predefinidas

- ARTIMÉTICAS: *+, -, *, /, remainder, quotient, sqrt, etc.*

PREDICADOS: Funciones que generan como resultado un valor booleano

- RELACIONALES: *<, <=, >, >=, =*
- LÓGICAS: *and, or, not*
- OTRAS: Manejo de listas...

Ejemplos
Expresiones aritméticas en Scheme

$$\frac{a+b}{2}$$

$$\frac{(5+a)*(b-8)}{c^3}$$

$$\frac{a}{2}+b$$

$$\frac{-b+\sqrt{b^2-4ac}}{2a}$$

Formas especiales

- Cada forma especial, tiene un orden de evaluación especial que la distingue (NO se utiliza el orden aplicativo).
- Formas especiales iniciales:
  - *define*
  - *quote*
  - *if*
  - *cond*

Forma especial *define*

- Sintaxis:

  (define símbolo expresión)
- Evalúa la expresión y el valor resultante lo asocia con el símbolo, de tal forma que el símbolo queda definido en el ambiente de trabajo.

Ejemplos

- (define a 5)
- (define b a)
- (define c (- a 3))
- (sqrt (- (* b b) (* 4 (* a c))))
- (< a b)
- (and (> a 0) (< b 0))
- (* a b c 4 953)
- (/ (+ a b) 2)
- (+ (/ a 2) b)
- (= (remainder c 2) 0)

Definición de funciones (procedimientos)

- Variante de la forma especial ***define***
- Sintaxis:

(define (nombre_función parámetros)
cuerpo)

Expresión que al evaluarse genera el resultado de la función

SÍMBOLOS
Los parámetros son sólo de entrada, y reciben el valor del argumento correspondiente.

Ejemplo

(define (cuadrado x) (x x))*

(define (area-circulo r)
(3.1416 (cuadrado r)))*

Forma especial *quote*

- Sintaxis:
  (quote dato)
- Produce una constante: invalida la evaluación del dato, generando como resultado el propio dato.
- Puede abreviarse con una comilla.
- *(quote abc)* es equivalente a *'abc*

Forma especial *if*

- Sintaxis:

  (if predicado exp-consecuente exp-alternativa)

- **Forma de evaluación**: Se evalúa el predicado, si su valor es verdadero se evalúa la exp-consecuente; si es falso, se evalúa la exp-alternativa (LAZY EVALUATION)

Forma especial *cond*

- Sintaxis:

  (cond
  (predicado$_1$ expresiones$_1$)
  (predicado$_2$ expresiones$_2$)
  …
  (else expresiones$_n$))

  Opcional

- **Forma de evaluación**: Evalúa las expresiones cuyo predicado sea verdadero, regresando el valor de la última; si ningún predicado se cumple, evalúa las expresiones del *else*.

Predicados predefinidos

- positive?
- negative?
- odd?
- even?
- integer?
- char?
- zero?
- …

Ejercicios

- Escribir un procedimiento que dados 3 valores numéricos, determine si se trata de un triángulo, en cuyo caso determine y regrese su tipo.

- Ejemplos de llamadas:

```
> (triangulo 3 6 8)   => escaleno
> (triangulo 6 6 6)   => equilatero
> (triangulo 6 8 8)   => isoceles
> (triangulo 3 3 8)   => no-triangulo
```