Paradigmas de programación

M.c Adan Garnica Carrillo

Universidad Michoacana de San Nicolás de Hidalgo

adan_0203@hotmail.com

01/09/2017

Contenido

Horario

- **Lunes 10:00 a 12:00** Salón FIE 8
- **Miércoles 11:00 a 12:00** Salón FIE 8
- **Duración del Curso** 48 Hrs.
- **Código con el curso en tiching.comm** 70QQK1

Contenido Sintético

Contenido	**N° de horas**
Historia y evolución de los lenguajes de programación	3
Lenguajes Imperativos	9
Lenguajes Orientados a Objetos	9
Lenguajes Funcionales	9
Lenguajes Lógicos	9
Lenguajes Script	9
Total de Horas	**48**

Bibliografía

- Friedman, D. P.; Wand, M.; Heynes, C. T., Essentials Of Programming Languages, The MIT Press, 1992
- Sethi, R., Programming Languages, Concepts and Constructs, Addison-Wesley Publishing Company, 1989

Bibliografía complementaría

- Budd, T., An Introduction To Object-Oriented Programming, Addison-Wesley Publishing Company, 1991
- Field, A. J.; Harrison, P. G., Functional Programming, Addison-Wesley Publishing Company, 1989
- Friedman, L. W., Comparative Programming Languages, Generalizing The Programming Function, Prentice Hall, Inc., 1991
- Kogge, P. M., The Architecture of Symbolic Computers, McGraw-Hill Incorporated, 1991
- Tucker, A. B., Jr., Lenguajes De Programación, 2ª Edición,

Evaluación

Aspecto	**Valor**
Asistencia	0%
Tareas	20%
Proyecto	20%
Practicas	20%
Exámenes	40%

Objetivo

Este curso introduce al estudiante a los diferentes tipos de lenguajes de programación contemporáneos. A lo largo de la historia de la computación, se ha desarrollado un gran número de lenguajes de programación, cada uno con diferentes objetivos en mente. Sin embargo, estos lenguajes se pueden clasificar, a groso modo, en cuatro familias: **Imperativos, Orientados a Objetos, Lógicos y Funcionales**. Dentro de este estudio, se incluyen los conceptos fundamentales que le proporcionan a un programador las herramientas necesarias para poder hacer un uso eficiente y con conocimiento de cualquier lenguaje de programación. Estos conceptos incluyen chequeo de tipos, administración de memoria, scoping, paso de parámetros, polimorfismo, etc., los cuales serán incluidos en los diversos temas, conforme se presenten en los diferentes lenguajes analizados. Esta estructura le proporciona al profesor la posibilidad de elegir los lenguajes con los cuales ilustrará los diferentes conceptos.

Introducción

Los lenguajes de programación han evolucionado conforme al avance de la tecnología y en consecuencia, han aparecido nuevos lenguajes, nuevas funcionalidades y nuevas formas de resolver los problemas. Es importante conocer los distintos enfoques de programación para elegir la herramienta que mejor nos convenga de acuerdo a los requerimientos de cada problema.

Programación Funcional **9 Horas**

1. Introducción
2. Objetos básicos (átomos y listas)
3. Evaluación de los átomos y de las listas
4. Definición de funciones (Anónimas y con nombres)
5. Predicados (Valores lógicos con tipos, de igualdad y operadores lógicos)
6. Estructuras de control (Condicionales e iteración)

Introducción a LISP

LISP fue inventado por John McCarthy en 1958 mientras el estaba en el MIT. McCarthy. La notación original usada por McCartyhy estaba basada en M-expressions. Esta fue rápidamente abandonada para convertirse en S-expressions, por ejemplo: la M-expression car[cons[A,B]] es equivalente a la S-expression (car (cons A B)). El nombre de LISP deriva de LISt ? Processing (procesamiento de listas). Se trata de una de las claves de este lenguaje de programación, las listas encadenadas.

Introducción a LISP

- Lenguaje Interpretado (También es posible compilarlo)
- Todas las variables son punteros
- Liberación automática de memoria (automatic garbage collection)
- Eficiencia menor que otros lenguajes a causa de que:
  - Es interpretado
  - Liberación automática de memoria.
- Adecuado para cálculo simbólico
- Fácil uso de la recursión

Introducción a LISP

Las listas son estructuras de datos dinámicas, esto es, su tamaño puede variar durante la ejecución del programa, lo que permite crear códigos dinámicos. Además son estructuras de tipo secuencial lo que quiere decir que para llegar a un elemento de la lista es necesario recorre primero todos los elementos previos y aunque esto se puede entender como una desventaja, se ha podido utilizar ésta característica para crear funciones que se aplican de manera secuencial a toda la lista en cuestión lo que hace más eficientes a los programas.

Introducción a LISP

El lenguaje de programación LISP fue uno de los primeros en ser usado para resolver problemas de inteligencia artificial, esto por ser un lenguaje de alto nivel que permitía el fácil manejo de estructuras de datos y de código, así como la creación de código dinámico. Aunque en la actualidad ya no es tan usado como antes, la ventajas que ofrece dicho lenguaje lo hacen muy atractivo al momento de atacar problemas del área de inteligencia artificial, donde el manejo de datos es trascendental, pensando en la optimización de los programas generados.

Objetos básicos átomos y listas

- Un átomo es un entero o un identificador.
- Una lista es un "(" un paréntesis izquierdo seguido de cero o más S-expressions, seguidos de un paréntesis derecho ")".
- Una S-expression es un átomo o una lista.

Ejemplo:
(A (B 3) (C) (( )))

Objetos básicos átomos y listas

- $< S - expression > ::= < atomo > \mid < lista >$
- $< atomo > ::= < numero > \mid < identificador >$
- $< lista > ::= (< S - expressions >)$
- $< S - expressions > ::= < vacio > \mid < S - expressions >< S - expression >$
- $< numero > ::= < digito > \mid < numero >< digito >$
- $< identificador > ::=$ Cadena o caracteres imprimibles (prohibidos los paréntesis)

Objetos básicos átomos y listas

- **NIL** es el nombre de una lista vacia
- En un test, **NIL** significa “falso”
- **T** es usado como ”true” pero cualquier cosa que no es **NIL** también es ”true”
- **NIL** puede ser un átomo o una lista

Salida formateada

{

- **A: Ascii.** Se imprime en formato ascii
- **S: S-expression.**
- **R: Radix.** **nR** imprime los valores númericos en el formato de la base especificada por ejemplo 10R es lo mismo que D.
- **C: Character.**
- **D: Decimal.**
- **B: Binary.** Imprime en formato base dos en lugar de base diez.
- **O: Octal.** Imprime en formato base 8 en lugar de base diez.
- **X: Hexadecimal.** Imprime en formato base Hexadecimal en lugar de base diez.
- **F: Fixed-format floating-point.** Se imprime en formato de punto flotante.
- **E Exponential floating-point.** Se imprime con notación exponencial.
- **G: General floating-point.** Se imprime en formato fijo, punto flotante o notación exponencial según convenga.

Evaluación de los átomos

- Evaluación de los átomos
- El valor de un número es el propio número.
- El valor de un símbolo es:
  - El número que tenga asignado, si actúa como variable numérica.
  - La S-expresión que tenga asignada, en caso contrario.
- El valor de una cadena de caracteres es la propia cadena.

Evaluación de las listas

- Las listas se interpretan como llamadas a funciones. El primer elemento es el nombre de la función y el resto son los argumentos. Por ejemplo, la lista (+ 2 3), se interpreta como la función + actuando sobre 2 y 3.
- Si queremos que la lista no sean evaluada como funciones debemos utilizar la función (QUOTE lista).
- Una alternativa para la función QUOTE es usar el apostrofe '.

Ejemplo: **(QUOTE (A B C D E)) o '(A B C D E)**

Definición de funciones

Una función es invocada como una lista donde el primer elemento es un símbolo (nombre de la función) y los elementos restantes son los argumentos.
Ejemplo: **(F A B)**

- **F** es el nombre de la función
- **A y B** son los argumentos

Sin embargo también los datos son escritos como átomos o listas:
Ejemplo: **(F A B)** es una lista de tres elementos.

Quoting

- (F A B) es una llamada a una función F, o es una lista?
- Los datos que no queramos que sean evaluados deben ser quoted
  - (QUOTE (F A B)) es una lista con (F A B)
    - QUOTE es una "forma especial" y los argumentos de una forma especial, no son evaluados.
  - '(F A B) es otra forma de realizar el QUOTING.

Quoting

- **CAR:** Regresa el primer elemento de una lista
- **CDR:** Regresa la lista sin el primer elemento
- **CONS:** inserta un nuevo encabezado en la lista
- **EQ:** compara dos átomos y regresa T si son iguales
- **ATOM:** Verifica si un elemento es un atomo
- **(NULL S):** Verifica si S es una lista vacia
- **(LISTP S):** Verifica si S es una lista
- **LIST:** Crea una lista con los elementos recibidos (evaluados)
  (LIST 'A '(B C) 'D) regresa (A (B C) D)
- **APPEND:** concatena dos listas
  (APPEND '(A B) '(X Y)) regresa (A B X Y)

FIN