

Tarea programada 3:Lisp

Base de datos de PDF's

Fernanda Fernández
Melissa Gutiérrez
Jean Carlo Alfaro

Junio, 2012

Table of Contents

Descripción del problema2

Diseño del programa3

Decisiones tomadas4

Estructuras de Datos5

Lógica del programa6

Manual de usuario8

 Instrucciones de ejecución8

 Ejecución de programa.....9

Conclusión.....10

Bibliografía10



Descripción del problema

El objetivo de esta tarea es familiarizarse con el desarrollo de aplicaciones usando el lenguaje Lisp, mediante la creación de una base de datos de información de archivos PDF's.

Deberán escribir funciones de Lisp que permitan leer archivos PDF, y decodificar la información presente. Para decodificar un archivo binario como un Pdf deberá escribir rutinas para poder leer archivos binarios.

Posteriormente, deberán crear una base de datos que se almacenará en memoria, la cual tendrá una estructura para poder almacenar la información de los pdf's que se encuentra como atributos.

Deberán tener una función que permita especificar un directorio de la computadora, y posteriormente que dicha función lea todos los pdf de dicho directorio, extraiga la información de los metadatos, y almacene esa información en la base de datos de memoria.

Se deberán crear funciones para poder hacer las siguientes consultas sobre la base de datos de los pdf's:

- Obtener todas las pdf de un determinado titulo
- Obtener todas las pdf de un determinado autor
- Obtener todas las pdf de un determinado fecha

Diseño del programa

Funciones:

- **(lectura “archivo-pdf”):**
Esta función recibe un pdf y luego extrae los metadatos
 - **(insertar-pdf objeto indice)**
Inserta un objeto pdf con los datos de metadatos en la estructura de datos que almacena los pdf(hash-table).
 - **(recorre(directorio))**
Basicamente recorre la dirección estipula por el usuario para buscar todos los documentos pdf 1.5
 - **(itera(tabla))**
Recibe como parámetro la tabla hash, en la cual utiliza sus valores para llamar a la función lectura.
 - **(buscar-titulo(str))**
Imprime los strings que cumplan con ser el título del documento
 - **(buscar-autor(str))**
Imprime los strings que cumplan con ser el autor del documento
 - **(buscar-creador(str))**
Imprime los strings que cumplan con ser el creador del documento
 - **(buscar-fecha(str))**
Imprime los strings que cumplan con ser el fecha del documento
 - **(buscar-productor(str))**
Imprime los strings que cumplan con ser el creador del documento
-

Decisiones tomadas

Base de Datos

La base de datos se carga en tiempo de ejecución mediante la función (recorre “directorio”).

Consultas

Después de cargar la base de datos en memoria el usuario puede utilizar las siguientes funciones para realizar las consultas:

- Obtener las canciones de un determinado titulo: (consulta-titulo)
 - (consulta-autor): obtener las canciones de un determinado autor
 - (consulta-fecha-creacion): obtener las canciones de un determinado fecha-creacion
 - (consulta-creador): obtener las canciones de un determinado creador
 - (consulta- palabras-clave) obtener las canciones de un determinado palabras-clave
-

Estructuras de Datos

El programa es diseñado según los requerimientos de las 5 consultas base. Para cada una de ellas se realizaron diferentes funciones que permiten consultar los datos solicitados por el usuario, para ello se requirió de estructuración de una clase principal pdf. Esta adapta seis atributos importantes, las cuales son: nombre, autor, titulo, fecha-creacion, creador y palabras-clave; en ellos se almacenaran los datos principales de las consultas requeridas y que serán manipuladas en una estructura de datos de tabla hash.

Veamos la definición de la clase principal:

```
(defclass pdf ()  
  ((nombre-archivo  
    :initarg :nombre-archivo  
    :initform ""))  
  (titulo :initarg :titulo :initform "")  
  (autor :initarg :autor :initform "")  
  (fecha-creacion :initarg :fecha-creacion :initform "")  
  (creador :initarg :creador :initform "")  
  (palabras-clave :initarg :palabras-clave :initform "")))
```

A partir de la definición anterior, se requiere de la estructura de almacenamiento de la tabla hash (*ht*) para almacenar los atributos del objeto, de esta forma llenada la tabla y poder realizar funciones de mapeo para iterar el contenido del mismo, esto se da el por medio de la siguiente instrucción:

```
(defun itera(tabla)  
  (loop for k being the hash-keys in tabla using(hash-value value)  
    do (lectura(gethash k tabla))))
```

Luego por medio de instrucciones de comparación de llaves la función devuelve el valor de la llave encontrada.

La tabla hash fue creada por medio de la siguiente instrucción:

```
(defparameter *ht* (make-hash-table))
```

Y su método para inserción de llaves y valores a la tabla es la siguiente instrucción:

```
(defun insertar-pdf(objeto indice)  
  (setf (gethash indice *objetos*) objeto))
```

Lógica del programa

La clase pdf () antes mencionada tendrá un conjunto de funciones de complemento para realizar las tres consultas base de carga de datos:

- (consulta-titulo)
- (consulta-autor)
- (consulta-fecha-creacion)
- (consulta-creador)
- (consulta- palabras-clave)

Primero recorre(directorio), que tendrá la función de solicitar un directorio donde se encontrará los archivos.pdf, esta función está conformada por un loop que itera sobre el directorio solicitado, esta lee todos los archivos de extensión .pdf y los concatena el directorio dado con el nombre del archivo y su extensión .pdf

Ejemplo: directorio = "home/fer/Escritorio",
archivo = "/*.pdf"
resultado= "home/fer/Escritorio/*.pdf"

Veamos la función en las siguientes líneas de código.

```
(defun recorre(directorio)
  (let((archivo 0))
    (dolist (file (directory (make-pathname :type "pdf" :name :wild :defaults directorio)))
      (setf (gethash archivo *ht*) file)
      (setf archivo (+ archivo 1)))))

(defun itera(tabla)
  (loop for k being the hash-keys in tabla using(hash-value value)
    do (lectura(gethash k tabla))))
```

El resultado de la concatenación del string es parametrizado sobre la función de pdf, de la cual este abre el archivo en formato binario con el directorio que se generó con la función itera(tabla).

Una vez leído el archivo en binario, la función inicializa variables: byte.

Posteriormente se realiza un ciclo de lectura de los 5000 bytes posteriores, pero va a ir concatenando los bytes que son necesarios, como etiquetas, nombre, autor, fecha, creador, palabras-clave, entre otras, por otra parte también excluirá muchos de los códigos innecesarios que posee el contenido del archivo.

Veamos la función en las siguientes líneas de código.

```
(defun lectura (file pos)
  (let ( (in (open file :element-type '(unsigned-byte 8))) (contador 0) nuevo-pdf)

    (when in
      (let ((str (make-string (file-length in))))
        (dotimes (i (file-length in))
          (setf (char str i) (code-char (read-byte in))))

        ;(format t "~a~%"(buscar-autor str))
        ;(format t "~a~%"(buscar-creador str))
        ;(format t "~a~%"(buscar-fecha str))
        ;(format t "~a~%"(buscar-titulo str))
        ;(format t "~a~%"(buscar-palabras-claves str))
        (setf (gethash pos *objetos*)(crear-objeto "nuevo-pdf" "" (buscar-titulo str) (buscar-autor str) (buscar-fecha str)
          (buscar-creador str) (buscar-palabras-claves str)) )))))
```

La función crear_objeto toma una lista generada de la función.

```
(defun crear-objeto(nuevo-pdf archivo titulo autor creador fecha claves)

  (defparameter nuevo-pdf (make-instance 'pdf))

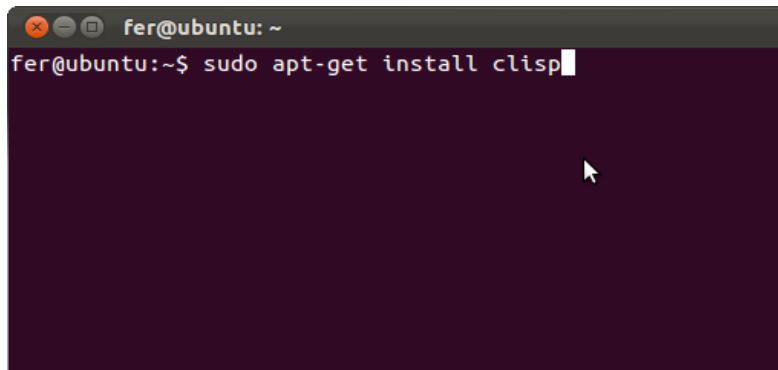
  (set-nombre-archivo archivo nuevo-pdf)
  (set-titulo titulo nuevo-pdf)
  (set-autor autor nuevo-pdf)
  (set-fecha-creacion fecha nuevo-pdf)
  (set-creador creador nuevo-pdf)
  (set-palabras-clave claves nuevo-pdf)

  (format t "~a~%"(get-nombre-archivo nuevo-pdf))
  (format t "~a~%"(get-titulo nuevo-pdf))
  (format t "~a~%"(get-autor nuevo-pdf))
  (format t "~a~%"(get-fecha-creacion nuevo-pdf))
  (format t "~a~%"(get-creador nuevo-pdf))
  (format t "~a~%"(get-palabras-clave nuevo-pdf))
  )
```

Manual de usuario

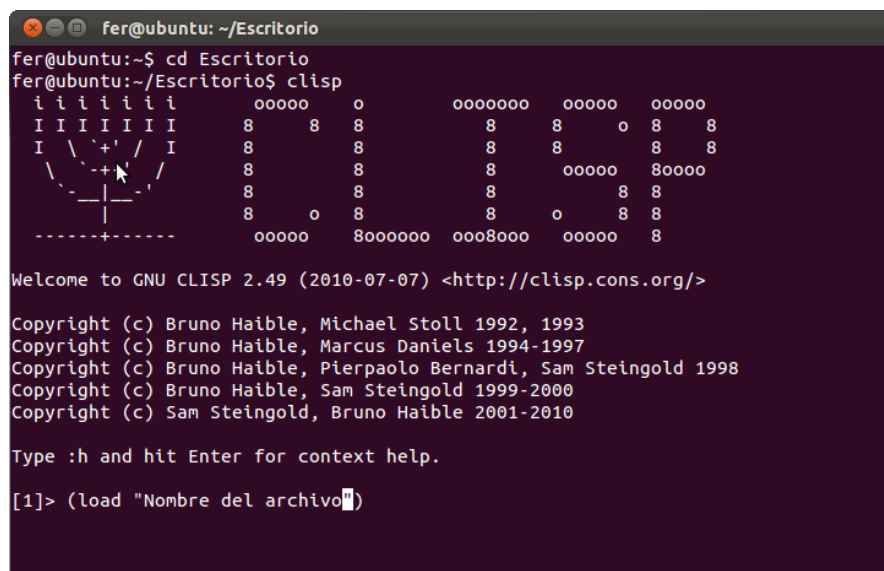
Instrucciones de ejecución

Primeramente deberíamos de instalar clisp desde terminal básicamente es escribir en terminal como se muestra en la figura.



```
fer@ubuntu: ~  
fer@ubuntu:~$ sudo apt-get install clisp
```

Posteriormente debemos cargar el escritorio por medio de: `cd Escritorio`, para poder llamar a los documentos que se encuentran en el mismo.



```
fer@ubuntu: ~/Escritorio  
fer@ubuntu:~$ cd Escritorio  
fer@ubuntu:~/Escritorio$ clisp  
      i i i i i i      00000 0      0000000 00000 00000  
      I I I I I I      8 8 8      8 8 8 8 8 8  
      I \ '+' / I      8 8      8 8 8 8 8  
      \ -+ - /      8 8      8 00000 80000  
      | | | | |      8 8      8 8 8  
      8 o 8      8 8 8 8 8  
      -----+----- 00000 8000000 0008000 00000 8  
  
Welcome to GNU CLISP 2.49 (2010-07-07) <http://clisp.cons.org/>  
  
Copyright (c) Bruno Haible, Michael Stoll 1992, 1993  
Copyright (c) Bruno Haible, Marcus Daniels 1994-1997  
Copyright (c) Bruno Haible, Pierpaolo Bernardi, Sam Steingold 1998  
Copyright (c) Bruno Haible, Sam Steingold 1999-2000  
Copyright (c) Sam Steingold, Bruno Haible 2001-2010  
  
Type :h and hit Enter for context help.  
[1]> (load "Nombre del archivo")
```

Luego por medio de `load` podemos cargar el `archivo.lisp` con el siguiente código (`load "nombre-archivo"`)

```
fer@ubuntu: ~/Escritorio
fer@ubuntu:~/Escritorio$ clisp
i i i i i i      ooooo  o      oooooooo  ooooo  ooooo
I I I I I I      8      8 8      8      8  o 8 8
I \ \ '+ ' / I   8      8      8      8      8 8
 \ \ -+ -' /     8      8      8      ooooo 8oooo
  \ | _ _ ' /     8      8      8      8 8
   | | _ _ ' /     8      o 8      8      o 8 8
  -----+-----  ooooo  8ooooooo  ooo8ooo  ooooo  8

Welcome to GNU CLISP 2.49 (2010-07-07) <http://clisp.cons.org/>

Copyright (c) Bruno Haible, Michael Stoll 1992, 1993
Copyright (c) Bruno Haible, Marcus Daniels 1994-1997
Copyright (c) Bruno Haible, Pierpaolo Bernardi, Sam Steingold 1998
Copyright (c) Bruno Haible, Sam Steingold 1999-2000
Copyright (c) Sam Steingold, Bruno Haible 2001-2010

Type :h and hit Enter for context help.

[1]> (load "TP")
;; Loading file /home/fer/Escritorio/TP.lisp ...
;; Loaded file /home/fer/Escritorio/TP.lisp
T
[2]> 
```

Ejecución de programa

1. Para ejecutar el código se debe digitar:
(load "TP3").
2. Y posteriormente se digita por el atributo que se quiera consultar,
ejemplo:
(consulta-titulo "Minuta #3")

Conclusión

La tercera tarea programada va muy acorde con los contenidos descritos en la carta al estudiante entregado al iniciar el curso.

Durante estas 3 semanas de curso, estudiamos el paradigma funcional, al cual pertenece el lenguaje de programación Lisp. Mismo que es utilizado para el desarrollo de esta tarea programada del curso.

Concluimos que se dificulta empezar una tarea programada sin previo estudio del proyecto como tal, por lo que se recomienda comprender en su totalidad el proyecto e ir detallando todas las partes del diseño para luego desarrollar el código fuente.

Se recomienda hacer un análisis profundo de los aspectos desconocidos en la especificación del proyecto es decir la función lamdba, implementación de listas y un amplio estudio de las estructuras, sintaxis en general de Lisp como lenguaje de programación.

Bibliografía

stackoverflow. (2011). Recuperado el 24 de Mayo de 2011, de <http://stackoverflow.com/questions/1403717/how-do-i-iterate-through-a-directory-in-common-lisp>

Apuntes de Common LISP. (s.f.). Recuperado el 23 de Mayo de 2011, de <http://lear.inforg.uniovi.es/ia/Archivos/Pr%C3%A1cticas/CommonLisp.pdf>

Common Lisp the Language, 2nd Edition . (s.f.). Recuperado el 23 de Mayo de 2011, de <http://www.cs.cmu.edu/Groups/AI/html/cltl/clm/node1.html>

Seibel, P. (2009). Recuperado el 23 de Mayo de 2011, de Practical Common Lisp: <http://gigamonkeys.com/book/>