Construcción Bases de Datos con Lisp.

Construction of Databases with Lisp.

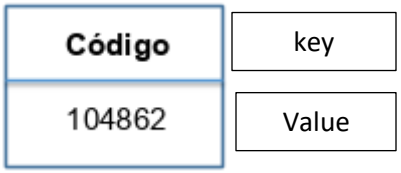
Juan David Osorio Ortiz
Jhonatan Ospina Osorio
Universidad Tecnológica de Pereira, Pereira, Colombia
Juandavid.osorio1@utp.edu.co

I. INTRODUCCIÓN

El presente trabajo se refiere a la creación de bases de datos con Lisp y la implementación de las funciones mas importantes para la creación de estas como el push, el select, el add, entre otras. La explicación e implementación de los códigos se encuentran todas en este paper con su debida explicación y código de ejemplo.

II. CONTENIDO

Para la introducción a la creación de bases de datos con Lisp debemos tener en cuenta un primer concepto que son las p-list o p-listas. La composición de esta estructura de datos está compuesta por dos "cajas", variables, en las que se almacenan el nombre del registro (llave) o en inglés 'key' y el contenido que este va a tener (valor) o value.



Para definir una p lista se puede hacer mediante el comando list, cambia en que cada uno de los ítems de la lista va a estar definido de la siguiente manera:

: (Field Name) (value)

Así se vería implementado en Lisp directamente:

```
;gnu clisp 2.49
(print (list :A 1 :B 2 :C 3 :D 4))
```

Esto seria lo que se imprime por pantalla una vez definida la lista:

```
(:A 1 :B 2 :C 3 :D 4)
```

GETF

Una de las ventajas del uso de las p-list es que se puede obtener los valores por el campo key con la funcion GETF (Get Field).

```
;gnu clisp 2.49
;DEFINICION DE LA LISTA EN UNA VARIABLE
(setq number (list :A 1 :B 2 :C 3 :D 4))
;SE BUSCA EN LA LISTA EL CAMPO LLAMADO ':A'
(print(getf number :A))
;SE BUSCA EN LA LISTA UN CAMPO QUE NO EXISTE
(print(getf number :E))
```

Como se puede observar en el ejemplo después de la definición de la lista se escribe el nombre de alguno de los campos a las que esta pertenezcan y como resultado nos dará el valor que este tiene asignado.
Así se demuestra en el siguiente ejemplo los resultados a la consulta anterior:

```
1
NIL
```

En el caso de que el campo no exista como resultado nos va a dar un NIL.

FUNCION CREACION REGISTRO

Para reducir la complejidad al momento de crear muchos registros de define una funcion que nos permita hacer esto generalmente en muchos casos, de la siguiente manera:

Fecha de Recepción: (Letra Times New Roman de 8 puntos)
Fecha de Aceptación: Dejar en blanco

```
;gnu clisp 2.49
;creacion de la funcion para hacer un
;registro de la informacion personal
;PN Primer nombre
;SN Segundo Nombre
;PA Primer Apellido
;SA Segundo Apellido
;CC Cedula
(defun MakeReg(PN SN PA SA CC)
    (list :PN PN :SN SN :PA PA :SA SA :Cedula CC)
    )
;Se crea un primer registro para probar la funcion
(setq Informacion (MakeReg "juan" "david" "osorio" "ortiz" 1088255))
;Se inprime el registro creado
(print Informacion)
```

Una vez definida la funcion se procede a probarse como quedaría la creación de un registro.

```
(:PN "juan" :SN "david" :PA "osorio" :SA "ortiz" :CEDULA 1088255)
```

CREACION DE LA BASE DE DATOS

Para la creación de la base de datos inicialmente se inicia por definir una variable global, que como se sabe se define de la siguiente manera:

(defvar *db* NIL)

En el comando anterior se define una variable de nombre db e inicializada como vacía, ya que aún no se ha agregado ningún registro.

Para introducir información dentro de una lista en Lisp es posible mediante el uso de la funcion push, a continuación, el ejemplo implementado para crear un registro y agregarlo con la información de una persona:

```
;Creacion de la variable que contiene la DB
(defvar *InformacionPersonal* nil)
;Funcion Que creaa el Registro
(defun MakeReg(PN SN PA SA CC)
    (list :PN PN :SN SN :PA PA :SA SA :Cedula CC)
    )
;Funcion  que se encarga de agregarlo a la base
(defun AddPerson(MK)
    (push MK *InformacionPersonal*)
    )
;Creacion y adicion de registros
(AddPerson (MakeReg "juan" "david" "osorio" "ortiz" 1088255))
(AddPerson (MakeReg "juan" "camilo" "jimenes" "castro" 103534))
;Se inprime la base de datos
(print *InformacionPersonal*)
```

En el anterior ejemplo se reutiliza la funcion creada para hacer el registro, y se define la nueva funcion que es la que agrega este registro a la base de datos. Y se agregan dos registros como ejemplo, En la siguiente imagen se muestra cómo se vería si imprimimos nuestra base de datos.

```
((:PN "juan" :SN "camilo" :PA "jimenes" :SA "castro" :CEDULA 103534)
 (:PN "juan" :SN "david" :PA "osorio" :SA "ortiz" :CEDULA 1088255))
```

Como se ve en la imagen anterior si se puede imprimir la base de datos, pero no es muy entendible. A continuación, se muestra cómo se imprime con un formato especifico.

MOSTRAR LA LISTA CON FORMATO

```
;Funcion que define el formato de impresion
(defun printDB()
    (dolist(MK *InformacionPersonal*)
            (format t "~{~a:~10t~a~%~}~%" MK)))
```

Con la funcion anterior se establece un formato para imprimir el texto por pantalla.

En la siguiente imagen se muestra el código completo y el resultado de la impresión.

```
;Creacion de la variable que contiene la DB
(defvar *InformacionPersonal* nil)
;Funcion Que creaa el Registro
(defun MakeReg(PN SN PA SA CC)
    (list :PN PN :SN SN :PA PA :SA SA :Cedula CC)
    )
;Funcion  que se encarga de agregarlo a la base
(defun AddPerson(MK)
    (push MK *InformacionPersonal*)
    )
;Funcion que define el formato de impresion
(defun printDB()
    (dolist(MK *InformacionPersonal*)
            (format t "~{~a:~10t~a~%~}~%" MK)))
;Creacion y adicion de registros
(AddPerson (MakeReg "juan" "david" "osorio" "ortiz" 1088255))
(AddPerson (MakeReg "juan" "camilo" "jimenes" "castro" 103534))
;Se inprime la base de datos
(printDB)
```

```
PN:         juan
SN:         camilo
PA:         jimenes
SA:         castro
CEDULA:     103534

PN:         juan
SN:         david
PA:         osorio
SA:         ortiz
CEDULA:     1088255
```

FUNCION REMOVE-IF-NOT O (SELECT)

```
(defun SelectByName(Name)
    (remove-if-not
        #'(lambda(MK)(equal(getf MK :PN) Name))
     *InformacionPersonal*))
```

Lo que hace esta funcion es que recibe como parámetro lo que queramos buscar, en la segunda línea se inicializa la funcion remove-if-not que lo que hace es que saca a una lista aparte todos los registros que cumplan con esta condición, en la 3 línea se define una funcion anónima, o lambda la cual hace uso de la funcion explicada anteriormente para sacar toda la información del campo que necesitemos y después compararlos con la información que recibimos por parámetro en la funcion, si son iguales los agrega a una lista aparte y estos son los que se imprimen al final, en las siguientes imágenes se ve el código completo con todo implementado.

```
;Creacion de la variable que contiene la DB
(defvar *InformacionPersonal* nil)
;Funcion Que creaa el Registro
(defun MakeReg(PN SN PA SA CC)
    (list :PN PN :SN SN :PA PA :SA SA :Cedula CC)
    )
;Funcion  que se encarga de agregarlo a la base
(defun AddPerson(MK)
    (push MK *InformacionPersonal*)
    )
;Funcion que define el formato de impresion
(defun printDB()
    (dolist(MK *InformacionPersonal*)
              (format t "~{~a:~10t~a~%~}~%" MK)))
;funcion select
(defun SelectByName(Name)
    (remove-if-not
        #'(lambda(MK)(equal(getf MK :PN) Name))
     *InformacionPersonal*))
;Creacion y adicion de registros
(AddPerson (MakeReg "juan" "david" "osorio" "ortiz" 1088255))
(AddPerson (MakeReg "carlos" "camilo" "jimenes" "castro" 103534))
;Se inprime la base de datos
(print(SelectByName "juan"))
```

Como vemos en la funcion de select pusimos que se imprimiera todos los registros con nombre juan. En la base de datos tenemos 2 registros y solo hay uno que cumple esta condición, esto es lo que se ve por pantalla.

```
((:PN "juan" :SN "david" :PA "osorio" :SA "ortiz" :CEDULA 1088255))
```

CONCLUSIONES

[1]. La implementación de estas funciones ayuda a entender mucho como es la estructura interna de estas mismas a la hora de entender otros lenguajes de programación que se basan en esto.

[2]. Gracias a la realización e implementación de estas funciones se pueden resolver muchos problemas y plantearse nuevas funciones para la implementación de bases de datos en Lisp.

1. Las notas de pie de página deberán estar en la página donde se citan. Letra Times New Roman de 8 puntos