

Q476.5
G6

INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE FISICA Y MATEMATICAS

**UN INTERPRETADOR LISP PARA UN
SISTEMA COMPUTADOR PDP-15**

**T E S I S
QUE PRESENTA
RAUL GONZALEZ NAVIDAD
PARA OBTENER EL TITULO DE
LICENCIADO EN FISICA Y MATEMATICAS**

MAESTRIA EN INGENIERIA

— — — — —

MEXICO, D.F.

1974

1 HIS BADGES

A MIS HERMANOS
DUENS, HECTOR, RAQUEL y ENRIQUE

Agradecemos al Dr. Harold V. McIntosh,
al Ing. Tomás A. Brody y a los Licen-
ciados Carlos García Jurado y José -
Urbano Cruz Gedille; su valiosa ayu-
da al asesorar este trabajo.

CONTENIDO

INTRODUCCION:	(1)	
CAPITULO I	CONCEPTOS BASICOS.	(3)
a)	Procesos Recursivos.	
b)	Estructuras de Listas.	
CAPITULO II	EL LENGUAJE LISP.	(12)
a)	Listas y Atomas.	
b)	Funciones Primitivas.	
c)	Pseudo-Funciones LISP.	
d)	Las Formas LAMBDA y LAMBDA*.	
e)	Definición y Evaluación de Estructuras de Listas.	
f)	Observaciones.	
CAPITULO III	IMPLEMENTACION DEL LENGUAJE LISP.	(20)
a)	Definición de la Filosofía para la Implementación.	
b)	Rutinas de Entrada y Salida (IO).	
c)	Funciones Primitivas.	
d)	Pseudo-Funciones y Formas LISP.	
e)	Algoritmo de Interpretación.	
f)	Colector de Basura.	
g)	Biblioteca.	

CAPITULO IV

GENERALIDADES

(56)

- a) Desarrollo Futuro.
- b) Aplicaciones.

CONCLUSION

(63)

APENDICE A: La Función DERIVADA y su Función auxiliar POLACA.

APENDICE B: Características del Sistema PDP-15.

APENDICE C: Programa (procesador LISP).

BIBLIOGRAFIA.

I N T R O D U C C I O N

LISP es un lenguaje de programación diseñado para procesar listas, característica de la cual deriva su nombre (LIST-PROCESSING).

Los aspectos más importantes de LISP son los siguientes:

- 1) Es un lenguaje matemático formal que está basado en la teoría de las funciones recursivas.
- 2) Está orientado hacia el procesamiento de datos simbólicos más que a datos numéricos.
- 3) Los programas escritos en LISP son a su vez listas.

LISP amplía el dominio de problemas a los cuales las computadoras pueden aplicarse, pues extiende las posibilidades de éstas hacia la manipulación simbólica, incluyendo una gran variedad de estructuras y la forma de procesarlas. Así, LISP como un lenguaje general de programación puede ser aplicado a una amplia gama de problemas de procesamiento numérico simbólico o mixto.

LISP específicamente ha sido aplicado a problemas en máquina (computadora) verificando demostraciones matemáticas, como lenguaje base para la implementación de otros lenguajes, en el cálculo diferencial-integral, teoría de circuitos eléctricos, simulación, lingüística, recuperación de in-

formación, gráficas, programación en pantalla y edición de -- textos.

Por lo tanto, la expansión de la naturaleza de los problemas solubles en un computador por medio de LISP parece no tener límite dada la diversidad de los problemas a los cuales puede aplicarse, lo cual lo hace un lenguaje de programación muy poderoso.

El propósito fundamental en la realización de este trabajo es el de poseer un procesador LISP con el cual las personas interesadas en familiarizarse con los procesos que LISP puede ejecutar, tengan en este trabajo la ayuda necesaria para aprender y practicar los conceptos básicos de LISP, así como introducirse a la forma en que este procesador se implementó en el sistema PDP-15.

CAPITULO 1
CONCEPTOS BASICOS

a.I) Procedimiento Recursivo.-

En casi todos los problemas que se programan para ser resueltos por una computadora se utilizan básicamente dos tipos de procesos: Procesos Iterativos y Procesos Recursivos.

Proceso Iterativo.-

Analicemos en detalle las características esenciales de un proceso iterativo.

a) Cada ciclo de iteración (conocido por el programador como un "LOOP") es ejecutado completamente antes de que la próxima iteración principie, y la decisión sobre si continúa iterando es tomada usualmente al final del ciclo usando los resultados obtenidos durante la ejecución corriente del mismo (o la decisión puede también haberse establecido antes de que el ciclo de iteración dé comienzo).

b) Los resultados obtenidos en cada ejecución del ciclo sirven como puntos de comienzo para el próximo ciclo, pero además estos valores pueden perderse; en la práctica los resultados de cada ejecución del ciclo se sobreescreiben sobre los anteriores (esta segunda característica no se aplica completamente en ciertos casos, por ejemplo en la multiplicación de matrices).

Consideremos unos ejemplos de procesos iterativos.

Ejemplo 1.-

Sea la función factorial FACT(N) y evaluemos el factorial de un número N>0. El proceso tendrá la siguiente forma:

FACT = 1 (cond. inicial)

FICT = FACT N

$$N = 11-1$$

N ? O

(cond. final, itera
6 sale del ciclo)

Cuerpo del ciclo.

Ejemplo 2.-

Sea la función multiplicación MULT(X Y) y evaluemos el producto de dos números X, Y. El proceso tendrá la siguiente forma:

MULT = 0 (cond. inicial)

15

(cond. final, itera
ó sale del ciclo)

Cuerpo del ciclo.

Digitized by Google

16

Proceso Recursivo.

Un proceso diferente que usa varias veces una misma parte de un programa es el llamado "recursivo".

Miremos la función factorial que está ahora definida en forma recursiva.

$$\text{FACT}(N) = \begin{cases} 1 & \text{si } N = 0 \\ \text{FACT}(N-1) * N & \text{de otra manera} \end{cases}$$

Veamos cómo trabaja este proceso recursivo para un valor $N > 1$.

$\text{FACT}(N)$ es calculado por medio de la fórmula de recursión; pero como $\text{FACT}(N-1)$ no es conocido ($N > 1$), el resultado parcial es almacenado y se procede a evaluar $\text{FACT}(N-1)$ con la misma fórmula de recursión, continuando de esta manera hasta que la condición terminal ($N=0$) es encontrada y los resultados parciales almacenados son recuperados (en orden inverso) y los cálculos son completados.

Las características de un procedimiento recursivo son:

- a) Cada ciclo de recursión (el cual será llamado - un nivel en lo que sigue) no se completa antes que el próximo es comenzado; frecuentemente una condición de recursión llamará al próximo nivel casi al comienzo de la ejecución del nivel.
- b) Resultados parcialmente completos de niveles anteriores deben estar almacenados de tal manera que ellos puedan ser recuperados de nuevo para completar los resultados -- cuando el proceso tome el sentido inverso.
- c) El punto en el cual el programa (o subprograma) que representa a la función recursiva fue dejado debe ser re-

cordado ya que se debe regresar el control de la ejecución al programa principal una vez que todos los niveles de recursión han sido completados (es decir, cuando el proceso recursivo haya terminado).

Veamos como quedaría definida la función multiplicación $MULT(X Y)$ en forma recursiva.

$$MULT(X Y) = \begin{cases} X & \text{si } Y = 1 \\ X + MULT(X Y-1) & \text{de otra manera} \end{cases}$$

Para resumir, se puede decir que en un procedimiento recursivo, todos los niveles son llamados hacia arriba hasta el máximo paso de recursión, pero su ejecución es completa da únicamente cuando todos los niveles son recuperados hasta el último.

Nota Importante:

La recursividad de funciones tiene la ventaja de proporcionar gran generalidad, tiempo atrás estos procesos resultaban lentos, debido a que las computadoras no poseían los mecanismos adecuados para procesar este tipo de funciones. En la actualidad la velocidad con que se llevan a cabo procesos recursivos depende fundamentalmente de la máquina usada y de la experiencia del programador en estos procesos.

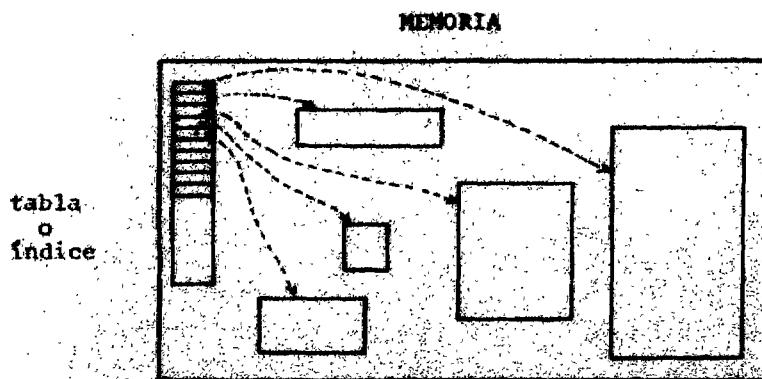
b.I) Estructura de Listas.-

Un problema a discutirse aquí es la presentación a la computadora de datos cuya estructura pueda ser más compleja que la de los números.

Aquí entra el concepto de Estructuras de Listas.

El concepto de Lista se originó del problema de mantener archivos voluminosos en orden, tan frecuente en programación comercial.

Si ciertos datos son almacenados, digamos en orden alfabético, una buena cantidad de tiempo de máquina se gastaría en mover los datos cuando éstos son borrados o nuevos son añadidos. Uno puede guardar los datos en forma individual sobre cualquier parte libre de la memoria, con tal que una tabla de direcciones (la cual capacita a la máquina a tener acceso a los datos) es construida, es decir, parte de la memoria se utiliza como un índice para la localización exacta de los datos (así descrito en la fig.)



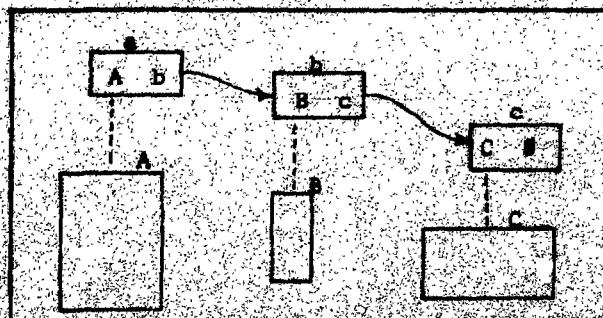
El problema con un índice de este tipo es que aún -
se tendría que mover la parte inferior del índice para introducir nuevos datos, y dejar espacios en blanco en el índice cuando se tuvieran que borrar datos o tener que dejar espacios en blanco a propósito entre las entradas al índice ya almacenadas para evitar mover el índice.

Ambas soluciones son malas porque en el primer caso la cantidad de información a moverse puede ser muy grande y - en el segundo caso el espacio en la memoria es usado muy inefficientemente.

Pero notese que lo importante en el índice no es la localización precisa de la próxima entrada, sino si tener una conexión entre entradas, así que el problema puede entonces manejarse por medio de lo que nosotros llamaremos una lista.

Una lista se obtiene así, si diseminamos las entradas de los índices en la memoria, colocándolos en cualquier parte pero añadiendo a cada entrada una segunda dirección (la cual es la dirección del próximo elemento sobre la lista) (—ver ilustración).

MEMORIA



- 6 -

Las pequeñas cajas con direcciones a, b, c, representan a los elementos de la lista y sus conexiones quedan representadas por las flechas continuas.

Cada elemento de la lista contiene:

- a) La dirección del objeto.
- b) La dirección del próximo elemento de la lista.

El último elemento de la lista contiene un terminador para indicar que no le sigue ningún otro elemento.

Hacemos notar que la estructura interna de los objetos puestos en orden por la lista no tiene nada que ver con la estructura de la lista. A estos objetos los llamaremos átomos. En la figura ellos están unidos a los elementos de la lista por flechas punteadas.

Una lista entonces queda formada por una sucesión de elementos que contienen la dirección de los objetos de la lista y la dirección del próximo elemento en la lista. Además, cuando tales listas estén sujetas a operaciones los objetos de la lista no deberán sufrir modificación alguna.

Un elemento de una lista puede él mismo ser una lista y en ese caso se tiene una ramificación o una lista no lineal, pues cuando ningún elemento de una lista es él mismo una lista, se dice que la lista es lineal.

NOTA:

Otros tipos de estructuras de listas son posibles. Por ejemplo, cuando los elementos de una lista apuntan a

dirección del elemento anterior a elios o cuando el último elemento de la lista en lugar de un indicador tiene un enlace con el primer elemento de la lista. Darían lugar a otros tipos de estructuras de listas, que aquí no se estudian porque son menos fáciles de manejar para cálculos simbólicos; pero tienen otras ventajas para otros fines.

El tipo de listas en las cuales ningún elemento contiene información acerca del elemento anterior es el que usa LISP dado que permite una simple y rápida manipulación, además de ciertas ventajas en la notación, pues este tipo de listas se puede representar en la muy simple y conveniente forma de paréntesis, la cual llamaremos notación de listas.

Por ejemplo:

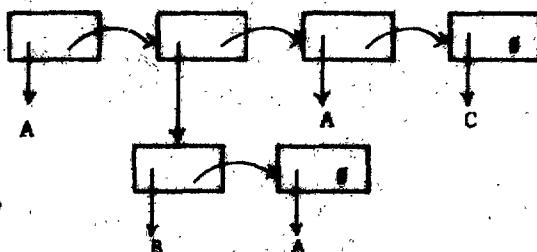
Una lista lineal de 3 elementos se describe así:

(A B C)

y una lista ramificada (es decir, cuando algunos elementos en la lista pueden ser a su vez listas,) se denotaría así:

(A (B A) A C)

Tal lista representa a la estructura en la siguiente figura:



Para usos posteriores es conveniente hacer notar la correspondencia biunívoca entre las notaciones list y los esquemas de listas (Kleene, 1952 p. 23).

CAPITULO II

EL LENGUAJE LISP

a.II) Listas y Atomas.-

Toda la notación de LISP está constituida por listas. Definimos técnicamente una lista como una serie de elementos (inclusive ninguno) encerrados entre paréntesis. Los elementos contenidos en una lista pueden ser a su vez listas o símbolos atómicos.

Definimos como un símbolo atómico a una sucesión de caracteres (los disponibles en una máquina de escribir) no interrumpida por paréntesis o espacio.

Ejemplo de símbolos atómicos:

ESCUELA FISICA

Ejemplo de lista:

((ESCUELA SUP. DE FISICA Y MATEMATICAS) () A)

b.II) Funciones Primitivas.-

Las funciones primitivas son cinco, tienen la característica de evaluar sus argumentos antes que ellas se evalúen cuando están en un proceso de evaluación.

- 13 -

(CAR L) Su valor es el primer elemento de la lista L. Si L = (A B C D), (CAR L) = A.

(CDR L) Su valor es la lista L sin su primer elemento. Si L = (A B C D), (CDR L) = (B C D).

NOTA:

El CAR y el CDR quedan indefinidos si su argumento resulta ser un átomo o una lista vacía. CDR siempre arroja una lista como su valor.

(CONS X Y) Y debe evaluar a una lista.

Su valor es una nueva lista construida con el valor de X y los elementos de Y.

Si X = (A B) y Y = (C D), (CONS X Y) = ((A B) C D)

Las dos funciones restantes son funciones predicado (es decir, su valor siempre es verdadero T o falso F.)

(ATOM L) Su valor es T si L evalúa a un átomo; F en otro caso.

(EQ L M) Su valor es T si L y M evalúan a átomos iguales; F en otro caso.

c.II) Pseudo-Funciones LISP.-

En LISP además de las 5 funciones primitivas defini

das, se tienen como parte integral de él otras funciones llamadas pseudo-funciones, debido al hecho de que ellas pueden no evaluar todos sus argumentos antes de que la función sea evaluada.

(QUOTE X) Su función es impedir la evaluación de su argumento, su valor es su argumento sin evaluar.
Si $X = (A B C D)$, $(QUOTE X) = X$.

La pseudo-función IF.

En la definición de una función LISP el proceso a seguir con los argumentos dependerá la mayoría de la veces de la naturaleza de éstos, por lo que es necesario probar de qué tipo son. Para hacer ésto usamos predicados y dividimos con ellos el conjunto de todos los posibles argumentos de la función en subconjuntos de argumentos para los cuales un predicado toma el valor T o F y para cada uno de estos resultados corresponderá aplicar una diferente función. La pseudo-función IF es una de las funciones que sirven para este propósito.

(IF P E₁ E₂) Esta función trabaja de la siguiente manera: Evalúa P, la cual debe ser una expresión predicado; si su valor es T se evalúa E₁; de lo contrario, evalúa E₂. El valor así obtenido es el resultado de toda la expresión.

d. II) Las Formas LAMBDA y LAMBDA*.-

Una forma es una expresión susceptible de arrojar o poseer un valor cuando sus variables sean identificadas y asociadas con valores.

El valor de una forma del tipo

((LAMBDA (V₁ V₂ . . . V_N) expresión) A₁ A₂ . . . A_N)

se calcula de la siguiente manera:

- a) Se evalúan A₁, A₂, . . . A_N.
- b) A cada V_i se le asocia el valor de la A_i correspondiente.
- c) Con estos valores asignados a las variables V_i, se calcula la expresión correspondiente.
- d) El valor de la expresión entera es el valor arrojado en la parte c.

La forma LAMBDA*:

((LAMBDA* (V₁ V₂ . . . V_N) expresión) A₁ A₂ . . . A_N)

a diferencia de LAMBDA, no evalúa sus argumentos, sino que asocia V₁ con A₁, V₂ con A₂, . . . , V_N con A_N. Entonces se procede igual que en la forma LAMBDA. Esto equivale a

((LAMBDA (V₁ . . . V_N) expresión) (QUOTE A₁) . . . (QUOTE A_N))

Existen dos casos especiales para las formas LAMBDA y LAMBDA*:

((LAMBDA L expresión) A₁ A₂ . . .)

Esta forma LISP se calcula de la siguiente manera:

- a) Se evalúan cada uno de los argumentos A_i .
- b) Se forma una lista con los valores calculados y es asociada a L como su valor.
- c) Con el valor asociado a la variable L se calcula la expresión correspondiente.
- d) El valor de la expresión entera es el valor arrojado en la parte c.

((LAMBDA* L expresión) $A_1 A_2 \dots$)

Igual a la anterior pero la lista asociada a L es la formada por los argumentos sin evaluar.

e.II) Definición y Evaluación de Estructuras de Listas.

Un programa en LISP consta de dos partes: una serie de definiciones puestas como argumentos de la función DEFINE y la instrucción para la aplicación de las funciones definidas a valores particulares de los argumentos por medio de las funciones EVAL o APPLY.

En LISP se puede asociar un nombre que es un símbolo atómico a una función construida con combinaciones de otras, de manera que cuando se quiera hacer uso de ella y ya se haya definido sólo será necesario poner su nombre seguido de sus argumentos, en lugar de poner toda la definición seguida de los argumentos.

DEFINE es la función que asocia un símbolo atómico

(el nombre de la función) a una definición. DEFINE tiene un número indefinido de argumentos, pero cada argumento es una lista cuyo primer elemento es el nombre de la función y el segundo la definición.

Estructura para DEFINE.

```
(DEFINE
  (NOMBRE1      E1)
  (NOMBRE2      E2)
  .
  :
  (NOMBREN      EN) )
```

en donde cada E_i es una expresión LAMBDA.

LISP permite la construcción de definiciones recursivas, tal característica es la más importante en el lenguaje.

Como un ejemplo de una función recursiva y su definición, consideremos una función de nombre ULTIMO; que tendrá un argumento y funcionará de la siguiente manera:

(ULTIMO X) X debe evaluar a una lista no vacía.

Su valor será el último elemento de la lista X.

Si X = (A B C), (ULTIMO X) = C

La función ULTIMO se construye y define de la siguiente manera:

```
(DEFINE (ULTIMO (LAMBDA (L) (IF
```

- 1º -

(NULL (CDR L)) (CAR L) (ULTIMO (CDR L)))

)

)

NOTA:

La función "NULL" incluida en esta definición es descrita a continuación en OBSERVACIONES.

La función EVAL es la instrucción para calcular el valor de una función para determinados argumentos dados. EVAL es una función de un argumento, el cual es una lista cuyo primer elemento es el nombre de la función que se quiere evaluar y los demás elementos de esa lista son los argumentos (inclusive ninguno) que necesita la función para ser evaluada.

(EVAL (NOMBRE ARG₁ ARG₂ ...))

)

APPLY tiene las mismas funciones que EVAL, pero difiere de ésta última en que mientras EVAL evalúa cada uno de los argumentos de la función antes de evaluar ésta APPLY suprime toda esa evaluación previa y evalua la función sin evaluar los argumentos.

(APPLY (NOMBRE A₁ A₂ ...)) es equivalente a

(EVAL (NOMBRE (QUOTE A₁) (QUOTE A₂) ...)).

OBSERVACIONES:

En LISP hay funciones incluidas entre las funcio--

nes de programación, de uso tan frecuente que se implementan como funciones de máquina, sobre todo para dar rapidez al cómputo.

Como ejemplos:

(NULL X) Su valor es T si X resulta ser la lista vacía; F de cualquier otra forma.

Si X = (A B C), (NULL X) = F

(PRINT X) Su valor es su argumento evaluado, pero como acción lateral imprime para el usuario el valor de X, PRINT tiene un uso frecuente cuando se trata de encontrar algún error dentro de una función.

Si X = (A B C), (PRINT X) = (A B C).

CAPITULO III
IMPLEMENTACION DEL LENGUAJE LISP

a.III) Filosofía de la Implementación.-

a) La implementación tendrá un carácter autogenerativo de funciones, es decir se definirá una serie de funciones de máquina lo más primitivas y básicas que sea posible y después con base en éstas se generarán todas las demás funciones LISP necesarias, aprovechando así las dos principales características de LISP de ser un lenguaje recursivo y tener capacidad de definir sus propias funciones.

b) Se adoptará la notación de lista descrita anteriormente, para mantener consistencia en el trabajo.

c) () es la lista vacía únicamente.

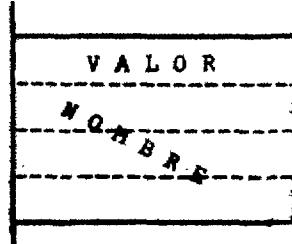
d) La función EVAL quedará implícita dentro de mecanismo de interpretación.

e) Los números serán considerados como átomos cualesquiera. Los caracteres especiales (\$ # ? etc.) serán considerados como caracteres comunes en la formación de los nombres de los átomos.

f) ATOMOS.

El formato para los átomos (manera de establecer estos en la memoria) constará de "valor" y "nombre", asignándosele una palabra de memoria para el valor y tres palabras para su nombre. Puesto que podemos almacenar tres caracteres por cada palabra de memoria, cada átomo tendrá un maximo de nueve caracteres en su nombre. Esta restricción en el tamaño del átomo da como resultado ganancia en velocidad durante la búsqueda de los mismos.

Formato para un átomo
(4 palabras)



g) MODOS.

En este trabajo un nodo es un par de palabras consecutivas de memoria que contienen apuntadores fundamentales en la representación de una estructura. Se necesitan dos palabras ya que cada palabra del sistema PDP-15 tiene 18 bits y puede almacenar cuando más un apuntador. El tamaño del nodo será pues de dos palabras y el formato (manera de establecer el CAR y el CDR) tendrá las siguientes características:

La primera de las dos palabras contendrá la dirección del CAR y la segunda la del CDR, permitiéndose para los direccionamientos 16 bits con los cuales se puede tener acceso a cualquier localidad de una máquina que tuviera hasta 64k pa-

laturas de memoria. Previsión que vale la pena hacer dado que el principal problema en LISP es su demanda de memoria.

Se establece además que CAR siempre estará sobre direcciones pares y por lo tanto CDR estará sobre impares.

Los dos bits restantes en cada palabra de cada nodo serán utilizados por el colector de basura y por el mecanismo de asociación de variables.

Formato para un nodo
(2 palabras)



h) RECURSIVIDAD Y ENLACE DE VARIABLES.

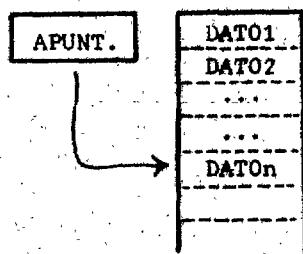
Tales problemas serán tratados por medio de un PUSH-DOWN-LIST(PDL).

Un PDL es un mecanismo de almacenamiento de datos, - que funciona de la siguiente manera: el último dato en entrar al PDL será el primero en salir.

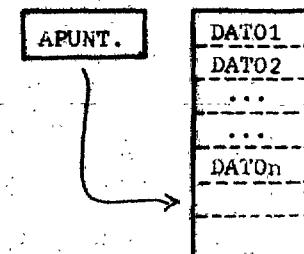
Un PDL consta de dos partes: Un apuntador y un área de almacenamiento.

Dos formas diferentes de trabajo para un PDL.

(1)



(2)



En la primera forma su apuntador hace referencia al último dato almacenado y no a la primera localidad libre.

En la segunda forma el apuntador refiere a la primera localidad libre y no al último dato almacenado.

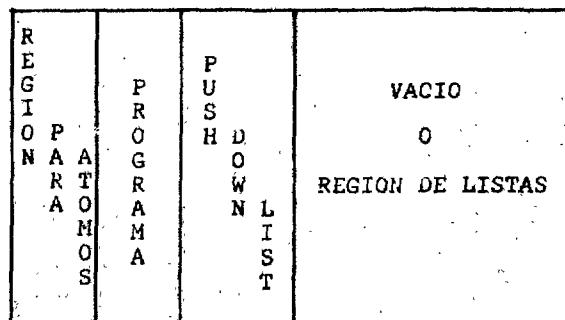
i) ESTRUCTURA

Se tendrá en la memoria una región fija para átomos una región fija para el PDL y una región fija para listas.

Dado que tenemos a disposición 24k de memoria, la distribución de ésta será como sigue:

2K	Región para átomos
2K	Programa
4K	PDL
16K	Vacio ó región de listas

Un esquema de la memoria sería el siguiente:



Tal distribución se ha establecido pensando con ella obtener ventajas en el direccionamiento de los distintos elementos que intervienen (nodos, átomos, PDL, etc.) y mejorar la velocidad de operación.

Así por ejemplo esta distribución nos permite usar la magnitud de las direcciones para distinguir un átomo de una estructura de lista y generar un criterio para detectar y diferenciar las funciones-máquina y las funciones-lista, dentro de las operaciones que lleva a cabo el sistema.

NOTA:

Funciones-Máquina. Todas aquellas funciones o formas LISP escritas en lenguaje de máquina (CAR, CDR, LAMBDA, --QUOTE, etc.).

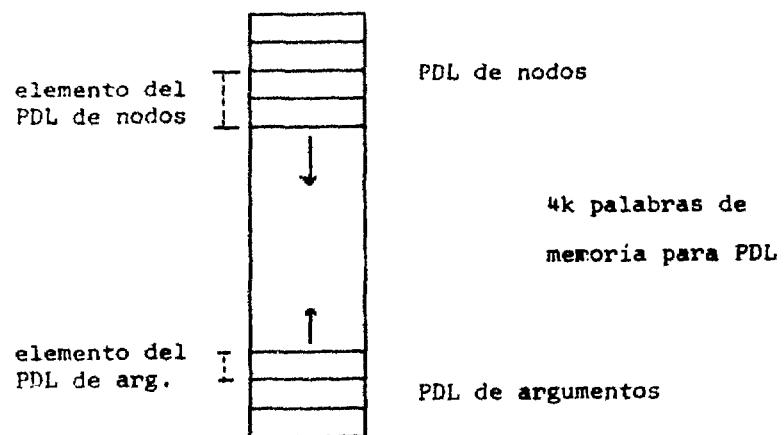
Funciones-Lista. Todas aquellas funciones definidas en un programa y que quedan grabadas en la memoria en forma de listas.

j) El resultado de toda función LISP es único y establecemos que ese resultado siempre queda como primer elemento disponible en el PDL de argumentos.

k) Como base para la implementación se tendrán dos mecanismos PDL actuando ambos en el área del PUSH-DOWN-LIST - descrita en la parte i).

1) Un PDL de nodos (es decir dos palabras por cada elemento del PDL), el cual será la base sobre la cual se construirá el mecanismo de interpretación.

2) Un PDL de argumentos (una palabra por cada elemento del PDL).



NOTA:

Es el PDL de nodos el mecanismo con el cual nos ayudaremos para llevar a cabo el enlace de variables y la recursividad de funciones.

- 1) Como inicialización al sistema de interpretación se tendrá una rutina que enlace las localidades de memoria, - dividiéndola en nodos enlazados formando una lista lineal de toda la memoria, con el objeto de dar a la rutina de lectura facilidades para que ésta establezca sobre la memoria las estructuras de listas.

- 2 -

b.III) Rutinas de Entrada y Salida (IO).

Es básico para la solución de un problema en una máquina computadora tener a disposición un conjunto de rutinas - que nos permitan proporcionar información a la misma y recuperar esta información una vez que ha sido procesada.

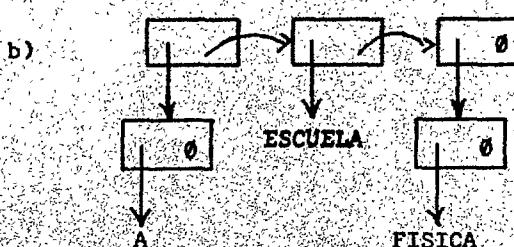
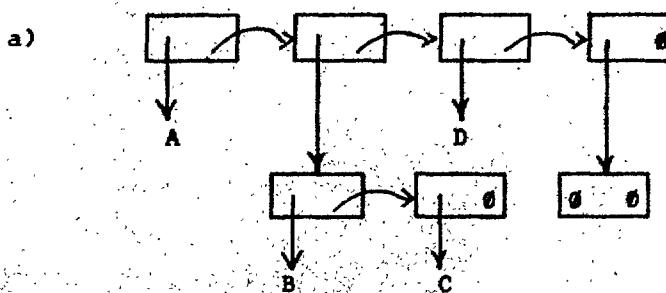
El problema para el caso especial de un intérprete - LISP se presenta de la siguiente manera:

Consideremos las siguientes expresiones

a) (A (B C) D ())

b) ((A) ESCUELA (FISICA))

En la memoria de la computadora estas expresiones -- quedan grabadas en la siguiente forma:



NOTA: En la figura los rectángulos representan nodos de la memoria y las flechas indican apuntadores o direcciones de memoria.

Al mecanismo que nos permitirá pasar las expresiones en notación de lista usual a expresiones equivalentes en la memoria de la máquina, será llamado rutina de entrada.

Al mecanismo que nos permita el proceso inverso será llamado rutina de salida.

Descripción de la Rutina de Entrada.-

El funcionamiento de esta rutina queda descrito por el diagrama a continuación:

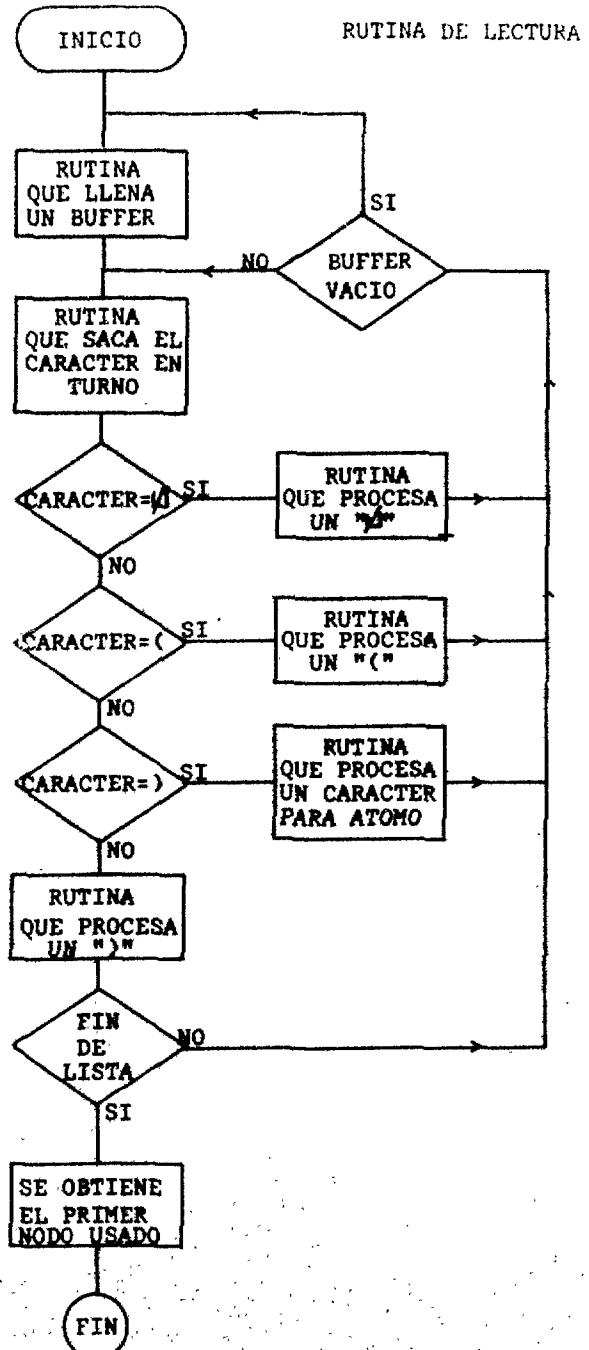
a) Rutina que llena un Buffer. Llena un buffer cuya capacidad es 72 caracteres, en el cual se hace un tratamiento especial a los siguientes caracteres:

RUBOUT.- Borra el último carácter introducido al buffer, reduciendo el contador de caracteres en 1.

CTRL U.- Borra todo el buffer, reduciendo el contador de caracteres a cero.

RT y LF.- Cierran el buffer colocando en éste una marca de fin de buffer, y se da por terminada la rutina que lo llena.

Cualquier otro carácter es permitido para formar parte del buffer. Si éste alcanza su capacidad máxima (de 72 caracteres) se procede de igual manera como si se procesara un RT o LF.



b) Rutina que saca el carácter en turno.

El planteamiento general es examinar los caracteres admitidos en el buffer de suerte que esta rutina pone a disposición de examen el carácter en turno en el buffer.

c) Rutina que procesa un carácter para átomo.

Los caracteres que son permitidos para formar átomos al pasar por esta rutina son empaquetados en un paquete que puede llegar a admitir hasta 9 caracteres (3 palabras por paquete) de modo que cadenas de más de 9 caracteres son cortadas a 9 ignorando el decimo en adelante, y el paquete así formado queda en espera de que se procese un carácter delimitador de átomos, como son los caracteres ")", ")", "(".

d) Rutina que procesa el carácter ")".

Pregunta si existe un paquete de átomos por cerrar. Si existe, lo cierra, lo busca y lo encuentra o lo coloca sobre la región de átomos, y lo procesa según la estructura de lista que se está leyendo. Si no existe ningún paquete el carácter se ignora.

e) Rutina que procesa el carácter "(".

Llama a la rutina que procesa el carácter ")" y además maneja los apuntadores que hacen encadenar las localidades de memoria para establecer en ésta la estructura de lista que se está leyendo.

- 10 -

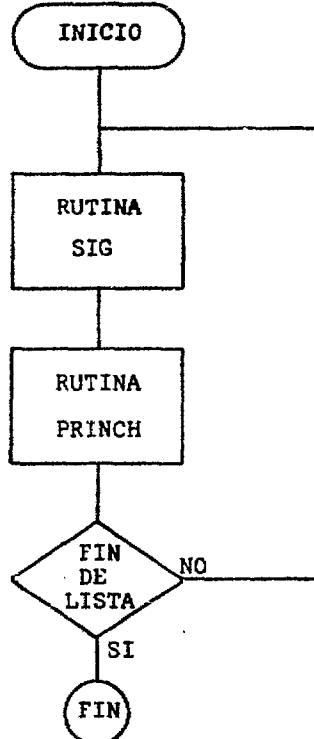
f) Rutina que procesa el carácter ")".

Trabaja en forma semejante a la rutina anterior, pero dejando una marca que indica el fin de una estructura cuando este carácter nivela los paréntesis en la estructura de lista.

Descripción de la Rutina de Salida

El siguiente diagrama ilustra la manera como fué -- construida la rutina de salida.

RUTINA DE SALIDA



Esta rutina escribe sobre el teletipo cualquier estructura de lista que esté grabada sobre la memoria; su único argumento es la dirección de la memoria en la cual la estructura a ser impresa da comienzo.

Descripción de la Rutina SIG.

Supongamos una estructura de lista grabada sobre la memoria, con la dirección de inicio (apuntador al primer nodo) para la estructura. La rutina SIG hace un rastreo de esta. Es decir, dado el nodo inicial SIG establecerá en sus diferentes registros información de ésta, indicando en ellos las posibles ramificaciones que pueda tener dentro de la memoria.

Registros especiales que usa la rutina SIG.

- S - Apuntador al siguiente elemento.
- L - Mitad izquierda de un nodo. -CAR-
- R - Mitad derecha de un nodo. -CDR-
- BI - Bandera de impresión.
- IR - Registro de índice.
- BAND - Bandera de inicio de estructura, 0 al inicio de la rutina de salida. (no la usa SIG).

Marcas especiales que utiliza la rutina SIG.

- Si L apunta a un átomo -- 0 en BI
- Si L apunta a una estructura-- 1 en BI
- Si se detecta fin de lista -- 2 en BI
- Si se detecta fin de estruc.-- 3 en BI

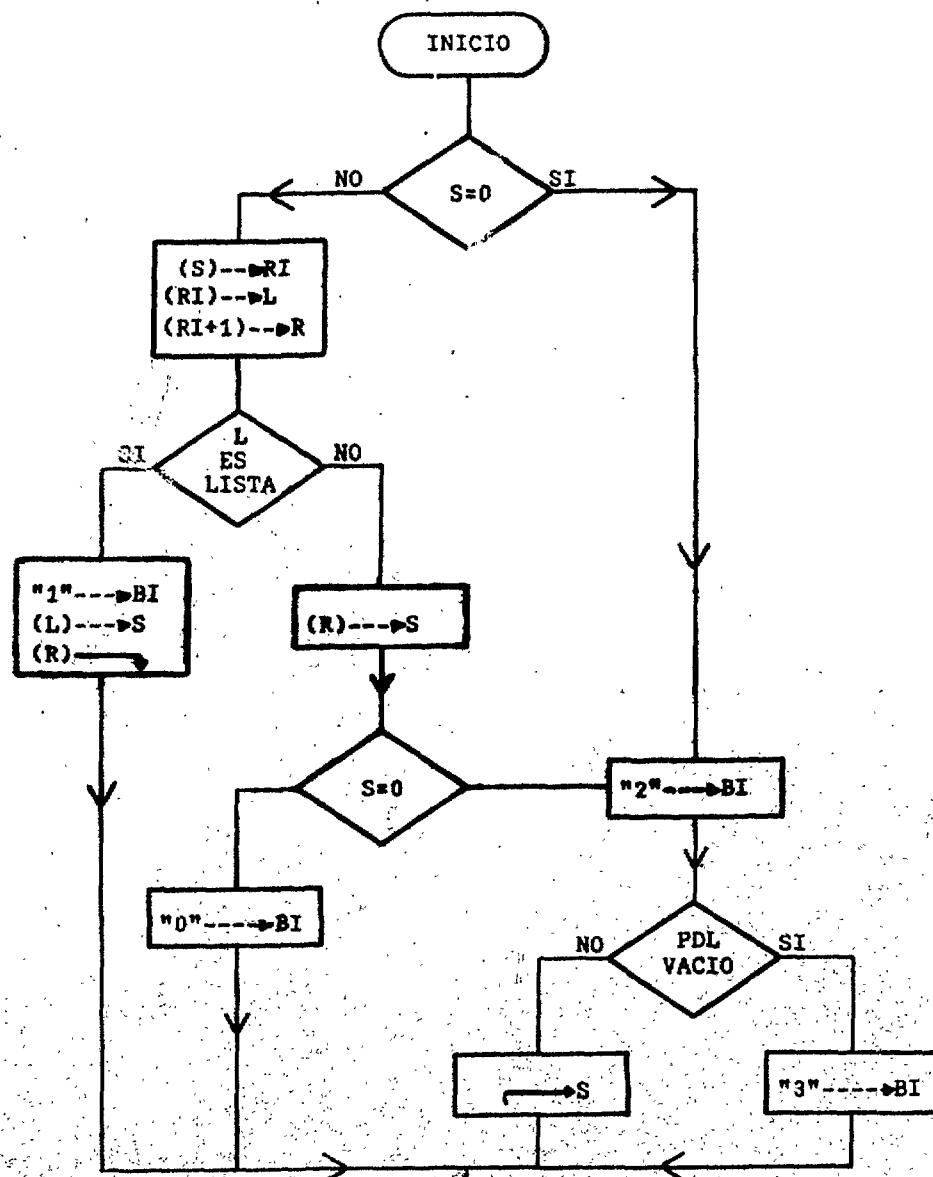
NOTA:

SIG será de gran utilidad cuando se construya el colector de basura. (ver sección correspondiente)

Descripción de la Rutina PRINCH

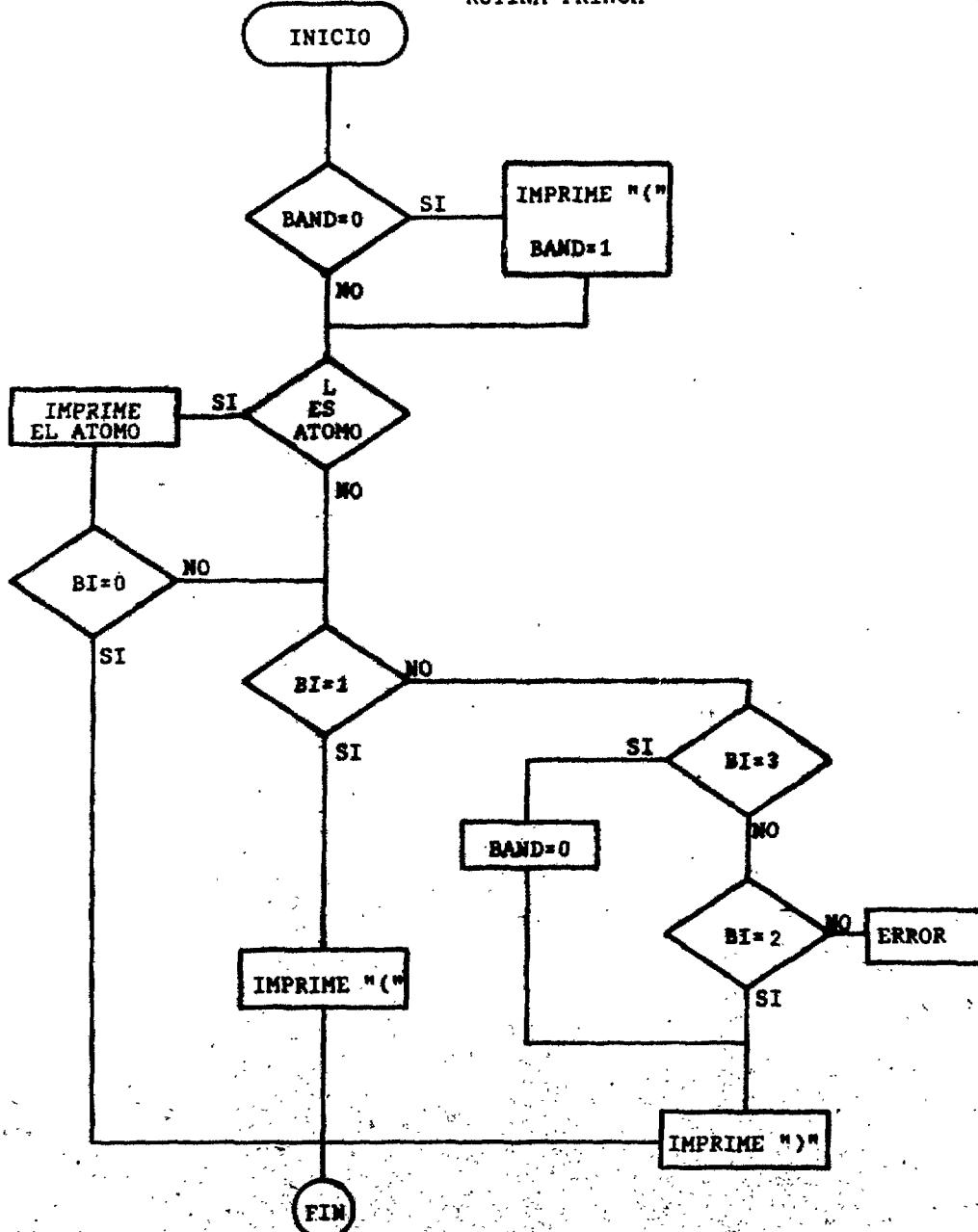
La rutina PRINCH imprime sobre el teletipo los símbolos (átomos, paréntesis, espacios, etc.) indicados por los valores que establece la rutina SIG.

RUTINA SIG



\rightarrow = contenido de X
 \rightarrow = introduce un valor al PDL
 \rightarrow = recupera un valor del PDL

RUTINA PRINCH



c.III) Funciones Primitivas.-

Una vez que se tienen estructuras de listas definidas sobre la memoria y la manera de imprimir éstas sobre el teletipo (Rutinas de entrada-salida), es necesario el diseño de los mecanismos que lleven a efecto los cálculos indicados por un programa LISP en ejecución.

Como una parte de este mecanismo general, consideremos la evaluación de las funciones primitivas (CAR, CDR, . . . etc.) cuando éstas son detectadas dentro de un proceso de evaluación.

En la evaluación de toda función LISP, las estructuras de listas que están definidas sobre la memoria no deben de sufrir modificación alguna, de tal manera que el efecto de evaluación de una función LISP debe ser únicamente la manipulación de los apuntadores a las direcciones de las estructuras establecidas en la memoria.

El proceso de evaluación es como sigue:

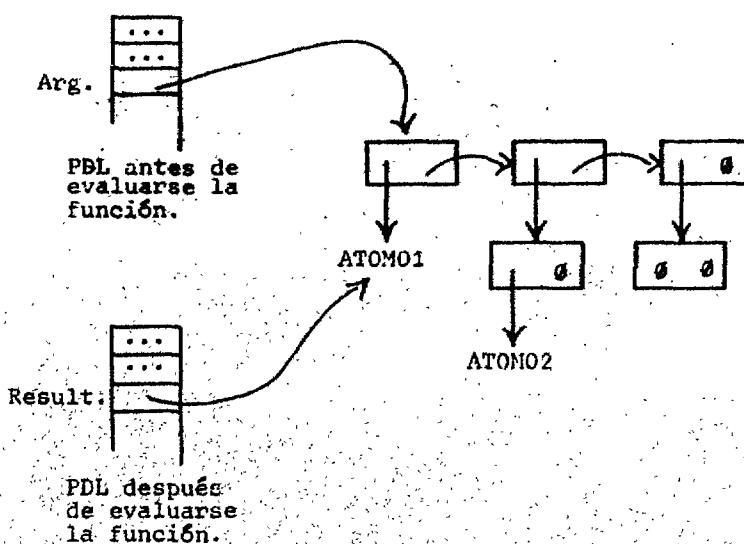
Se evalúan sus argumentos primero; tales argumentos quedan almacenados dentro de un PDL especial para argumentos y se procederá entonces a evaluar la función LISP en turno, la cual toma sus argumentos del PDL y una vez evaluada, su resultado será depositado en ese PDL especial para argumentos. Así, establecemos que toda función LISP actúa con el PDL de argumentos únicamente cuando se lleva a cabo el proceso de evaluación. Esta última característica es necesaria ya que en

la mayoría de los casos el resultado de la evaluación de una función LISP resulta ser argumento en la evaluación de otra función LISP.

Supondremos que los argumentos de una función LISP ya han sido evaluados y en esas condiciones enfocaremos el caso de llevar a efecto la evaluación de una función LISP.

Condiciones normales en el PDL de argumentos y del vacío o espacio para estructuras de listas cuando se detecta la ejecución de una función primitiva.

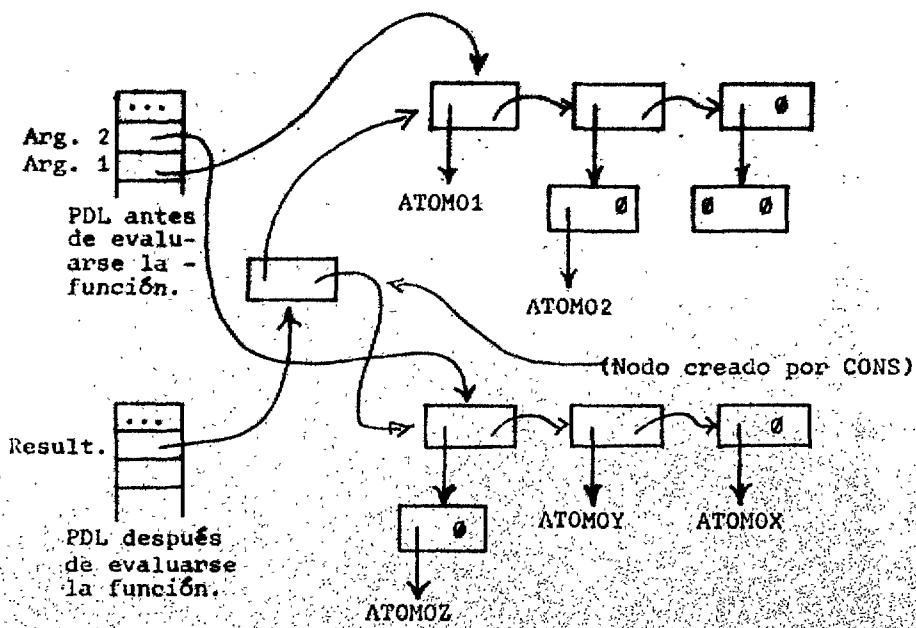
El mecanismo de evaluación de la función CAR es el siguiente; y el diagrama de memoria correspondiente se vería así en la evaluación de la función CAR.



Se extrae del PDL el primer elemento a disposición, el cual debe ser su argumento ya evaluado y por lo tanto debe ser un apuntador a una estructura de lista no vacía (debido a que CAR únicamente está definido cuando su argumento es una lista). Se obtiene el apuntador colocado en la parte izquierda del nodo apuntado y ese valor deberá ser colocado en el --PDL de argumentos. Esto termina el proceso de evaluación del CAR.

El mecanismo de evaluación de la función CDR se diferencia al de la función CAR, en que CDR obtiene la parte derecha del nodo apuntado en lugar de la parte izquierda.

El mecanismo de evaluación de la función CONS es el siguiente; y su diagrama de memoria correspondiente se vería así en la evaluación de la función CONS.



Se extrae del PDL el primer elemento a disposición, el cual debe ser el segundo argumento evaluado del CONS; este valor es colocado en la parte derecha de un nuevo nodo; se extrae del PDL otro elemento que será el primer argumento evaluado de CONS y es colocado en la parte izquierda de ese mismo nuevo nodo. El resultado total de la evaluación de CONS será colocar sobre el PDL un apuntador que refiera al nodo nuevo - utilizado.

Otras funciones como EQ, ATOM, NULL, etc. tienen un tratamiento semejante.

d.III) Pseudo-Funciones y Formas LISP.

En la construcción del mecanismo general de evaluación se plantea como un caso especial la evaluación de pseudo-funciones y formas LISP.

De manera semejante como en la evaluación de funciones LISP, las estructuras establecidas en la memoria deberán permanecer invariantes cuando se efectúe la evaluación de una pseudo-función o forma LISP. La relación entre estas estructuras y sus argumentos se mantendrá exclusiva sobre el PDL de argumentos.

Como es sabido, estas estructuras LISP bien pueden evaluar a todos, alguno o ninguno de sus argumentos, según sea la estructura tratada y según se comporten sus argumentos. Debido a esta última característica, cuando el proceso

de evaluación detecta una pseudo-función o forma LISP, éste ejecuta una transferencia de control al mecanismo que evaluará esa pseudo-función o forma LISP antes que sus argumentos se evalúen, y el hecho de la posible evaluación de sus argumentos hace que estos mecanismos tengan que prever la evaluación de éstos, interaccionando así con otros mecanismos de evaluación. Estos hechos establecen la diferencia de evaluación entre funciones y pseudo-funciones o formas LISP.

El mecanismo de evaluación de la pseudo-función QUOTE trabaja de la siguiente manera:

Como el resultado de la evaluación de la pseudo-función QUOTE es su argumento sin evaluar, el mecanismo QUOTE de evaluación simplemente coloca el apuntador del argumento de QUOTE como primer elemento disponible del PDL de argumentos.

El mecanismo de evaluación de la pseudo-función IF trabaja de la siguiente manera:

La pseudo-función IF tiene tres argumentos y el resultado de la evaluación del primero indica cuál de los dos restantes argumentos se evaluará y quedará como resultado de toda la expresión IF.

El mecanismo de evaluación de la pseudo-función IF coloca sobre el mecanismo general de evaluación las indicaciones necesarias para que éste evalúe su primer argumento y una indicación para que el mecanismo general regrese el control al mecanismo IF una vez que lo primero haya sido efectuado; --

en este punto se entrega el control al mecanismo general de evaluación. Así, una vez que se haya evaluado el primer argumento y el control lo tenga de nueva cuenta, el mecanismo de evaluación IF extrae del PDL de argumentos el primer elemento disponible (que será el resultado de la evaluación del primer argumento de IF), lo analiza pues ese valor debe ser un apuntador a los atomos T 6 F, y el resultado del análisis de ese apuntador dará la clave para indicar al mecanismo general de evaluación que actúe sobre el segundo argumento o sobre el -- tercero.

Otras pseudo-funciones y formas LISP tienen un tratamiento semejante de acuerdo a sus funciones y su descripción será omitida.

e.III) Funcionamiento del Mecanismo de Interpretación.-

Debido a la naturaleza recursiva de las funciones LISP, es necesario un mecanismo que almacene y recupere los valores que puedan adquirir las variables implicadas en una función cuando un proceso de evaluación se lleve a cabo sobre ella; esto es, una vez detectada la evaluación de una función, se hace necesario almacenar los valores de las variables implicadas en la definición de la función antes que ésta comience a evaluarse, y además se deberán almacenar los valores que se vayan adquiriendo conforme la evaluación de la función se lleve a cabo hasta su último nivel. Es entonces cuando se precisa recuperar los valores almacenados para completar la evaluación y una vez que la función en cuestión haya sido evaluada se procederá a restablecer las variables con los valores originales que éstas poseían antes de evaluarse la función.

El problema aquí tratado se soluciona con la siguiente estructuración.

Se establece que una vez que ciertos valores han sido asignados a cierta variable, sobre ésta siempre se mantendrá el último valor adquirido (valor corriente), así todo proceso de evaluación hará referencia a la tabla de átomos para obtener valores de variables. Los valores que durante el proceso hayan tenido que ser almacenados (valores viejos) quedan en el PDL de nodos con el siguiente formato:

en este PDL esta el resultado de una evaluación de alguna función LISP, y el intérprete mandará imprimir este resultado mediante la rutina de salida, quedando este PDL vacío tambien. - En ese instante se transfiere el control a la rutina de entrada, cuyo trabajo es establecer sobre la memoria alguna estructura de lista introducida mediante el dispositivo de lectura - y depositar el primer nodo usado en ello sobre el PDL de nodos; luego el intérprete procederá a revisar de nueva cuenta el PDL de nodos (que ahora no es vacío), se saca el nodo en turno del PDL, se analiza su contenido y de acuerdo al resultado de este análisis se definirá el proceso a seguir por el interpretador.

Por ejemplo:

Si el CAR del nodo en examen tiene la dirección de - un átomo (es decir CAR apunta a un átomo), se procede a investigar que tipo de estructura representa este átomo: función primitiva, forma LISP, función programada o simplemente una variable para una función LISP. Y en cada uno de estos casos se procede a hacer lo correspondiente para llevar a cabo su interpretación, para lo cual hace uso de los mecanismos antes descritos.

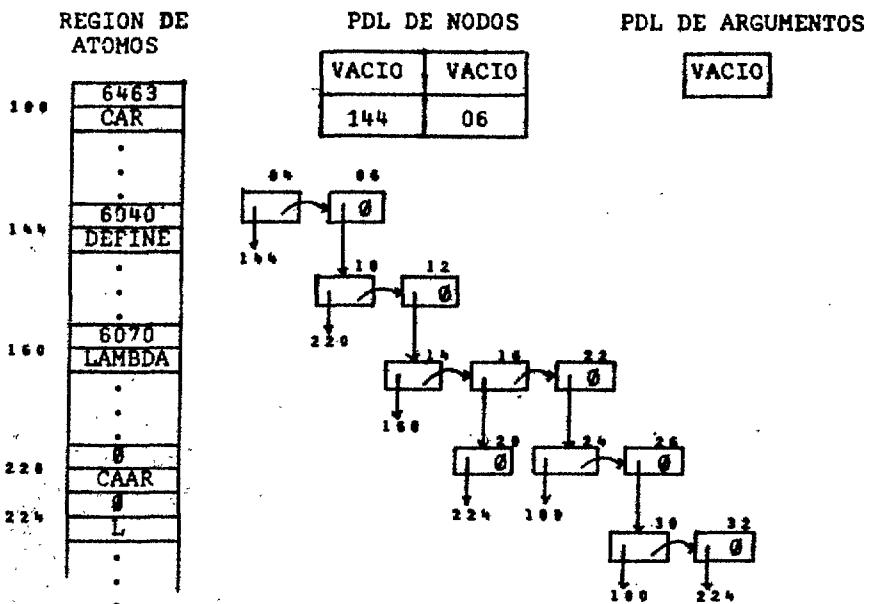
Describiremos el comportamiento de los mecanismos -- PDL y la región de vacío cuando se lleva a cabo un proceso de interpretación.

Supongamos que el sistema está esperando una estructura en la rutina de entrada y que el primer nodo libre es (04) y que la estructura por leerse es la siguiente:

(DEFINE (CAAR (LAMBDA (L) (CAR (CAR L)))))

Esta estructura es la definición en LISP de una función que evalúa sucesivamente dos veces la función CAR.

Al leerse la estructura se establece un parámetro -- sobre el PDL de nodos y ésta queda grabada sobre la región de vacío, como se ilustra en el esquema:



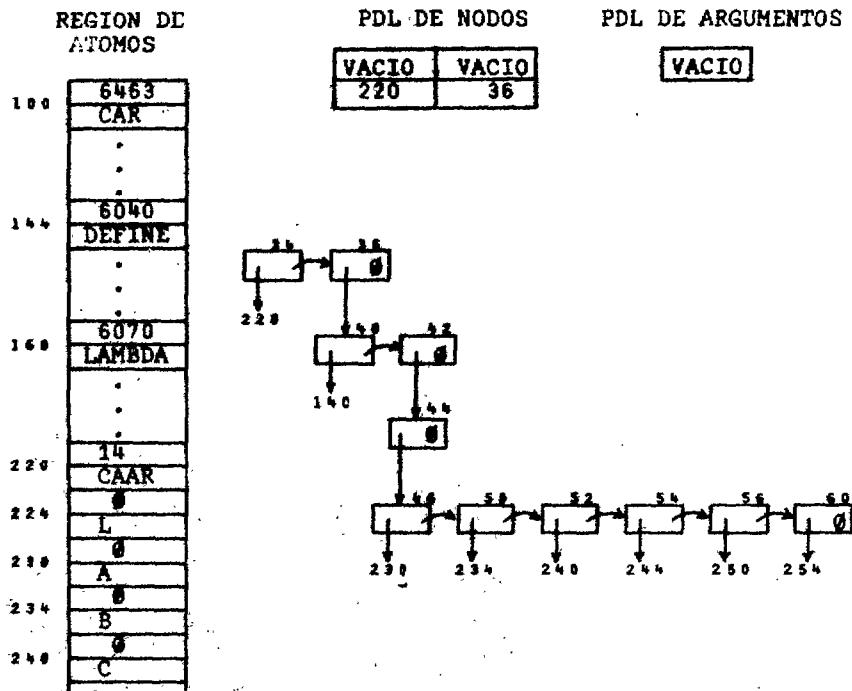
Una vez establecidas estas estructuras, el intérprete analiza el nodo en turno y descubre la indicación para que se ejecute la función DEFINE al detectar su dirección; entonces se cede el control a la rutina que ejecuta la función -

DEFINE, cuyo trabajo es colocar como valor para el átomo CAAR - la dirección (14), con lo cual el átomo CAAR queda definido como el nombre de una función LISP. Como resultado de esta ejecución el PDL de nodos queda vacío, el PDL de argumentos se mantiene vacío, así que al regresarse el control al interpretador éste manda directamente el control a la rutina de entrada. En este punto se puede volver a definir otra función o mandar ejecutar alguna función definida.

Supongamos que mandamos evaluar la función CAAR.

(CAAR (QUOTE ((A B C D E F))))

Después de leer esta estructura, la situación en memoria para el interpretador se vería de la siguiente manera:



Una vez que queda establecida esta estructura, el intérprete detecta la ejecución de la función CAAR y al interpretar las indicaciones de la función CAAR se llevan a cabo una serie de interacciones con los mecanismos PDL.

La situación de los mecanismos PDL en un paso intermedio durante la evaluación de la función CAAR es:

	PDL DE NODOS	PDL DE ARGUMENTOS
valor viejo	VACIO	VACIO
	200224	8
	100	0
	100	0

Situación de los mecanismos PDL cuando se ha terminado el proceso de evaluación de la función CAAR.

	PDL DE NODOS	PDL DE ARGUMENTOS
	VACIO	VACIO

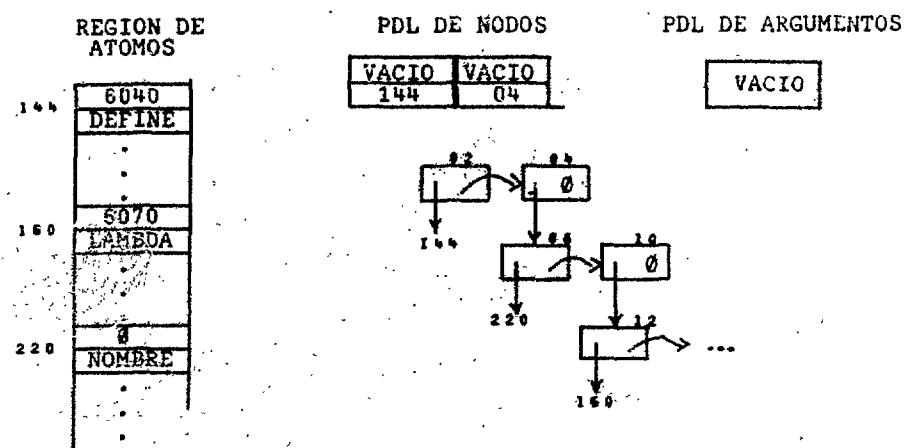
- 16 -

f.III) Colector de Basura.-

Consideremos la estructura en la memoria cuando se lleva a cabo un proceso de definición para una función.

(DEFINE (NOMBRE (LAMBDA ...

La estructura asociada antes de la ejecución de ---
DEFINE es la siguiente:



La estructura asociada después de la ejecución de --
DEFINE es:



DIRECCION	VALOR
-----------	-------

bit uno encendido indica valor viejo en el nodo

Así, una vez que el proceso de evaluación detecte un valor viejo en el PDL (para hacerlo corriente), éste sabrá a qué variable está asignado el valor mediante su dirección adjunta almacenada.

Hasta aquí se han descrito como partes aisladas las rutinas de entrada-salida, los mecanismos de evaluación de funciones primitivas y formas LISP, y la manera como los valores de las variables implicadas en una evaluación pueden mantenerse enlazados. Ahora describiremos cómo funciona el mecanismo de interpretación al conjuntar todas estas fracciones descritas.

El funcionamiento del mecanismo de interpretación se basa en un mecanismo PDL de nodos que funciona como un archivo en donde se depositan todas las indicaciones que debe seguir el intérprete al procesar alguna estructura de lista.

El funcionamiento del intérprete es el siguiente:

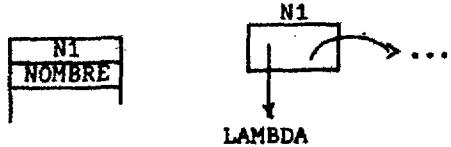
Primero revisa el PDL de nodos, si éste está vacío - no existen cálculos pendientes por hacer y por tanto pasará a revisar el PDL de argumentos, si éste no está vacío, entonces

Como se puede observar, una vez que `DEFINE` se ejecutó, no hay acceso (un apuntador) al bloque formado por los primeros cuatro nodos utilizados al establecer sobre la memoria la estructura, así estos cuatro nodos quedan desligados de cualquier estructura.

Supongamos ahora que queremos cambiar la definición de una función definida con anterioridad.

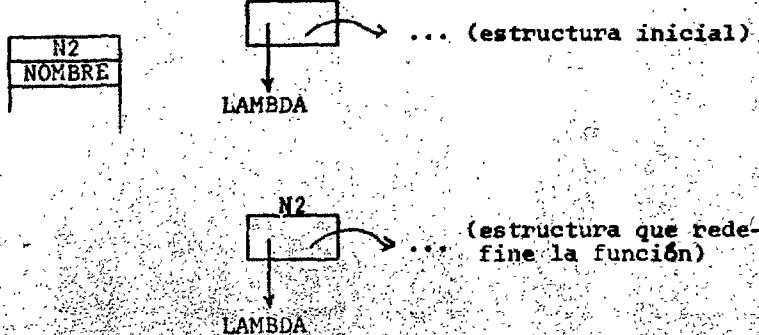
SITUACION INICIAL

REGION DE ATOMOS



SITUACION DESPUES DE LA REDEFINICION

REGION DE ATOMOS



En este caso se observa que la estructura inicial -- que definía la función queda sin acceso posible desde cualquier estructura al efectuarse la redefinición.

Este tipo de situaciones en que las estructuras establecidas sobre la memoria llegan a quedar completamente inaccesibles se presenta con mucha frecuencia en LISP. Así, en un momento dado pueden existir estructuras de este tipo que ocupen lugar en la memoria y a las cuales no se tenga acceso.

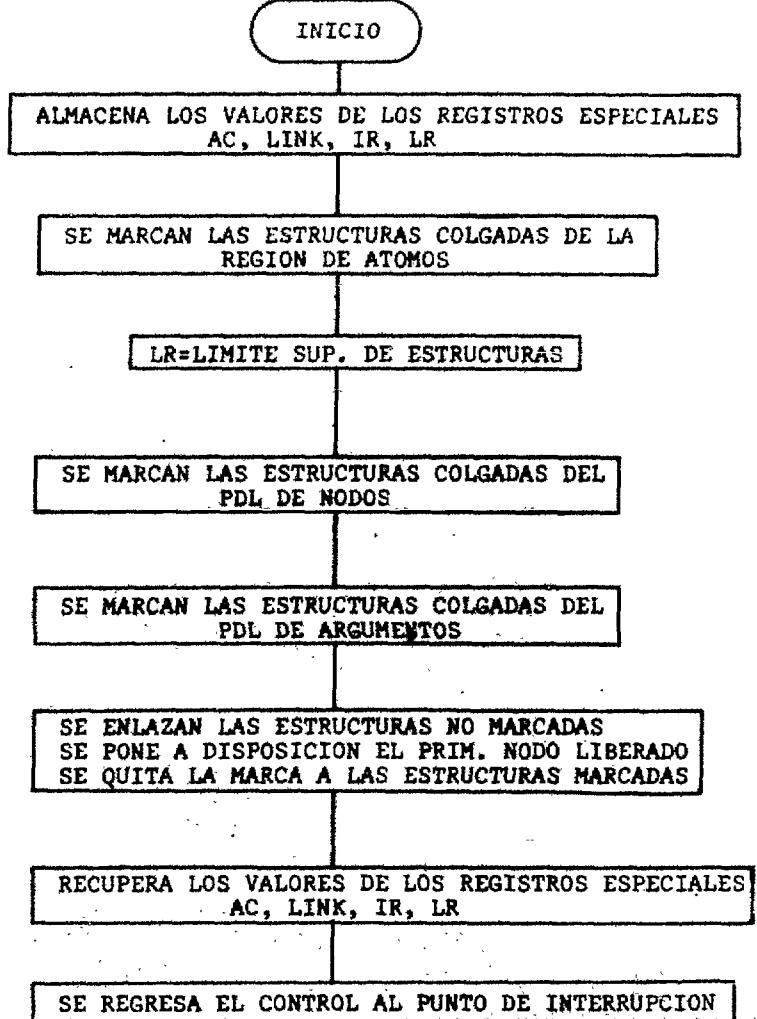
A este tipo de estructuras las llamaremos "basura".

Una vez establecida la posible existencia de basura, se establece tambien la necesidad de contar con un mecanismo que detecte y recolecte la basura y además ponga a disposición el área de memoria liberada. Este mecanismo al que llamaremos colector de basura trabaja bajo el siguiente principio:

Con la ayuda de la rutina SIG y de un bit por cada nodo de estructura para marcar con este todas las estructuras útiles (estructuras de memoria a las cuales se puede tener acceso), y enlaza formando una lista lineal todos aquellos nodos no marcados (basura), haciendo los útiles de nueva cuenta para futuros requerimientos de memoria.

El colector de basura es llamado automaticamente por el sistema interpretador cada vez que hay requerimiento de nodos libres y la región de vacío se ha llenado de estructuras de listas.

COLECTOR DE BASURA



g.III) Biblioteca.

Una de las características más importantes en LISP es su capacidad para definir y usar sus propias funciones. - Esta característica es la que establece la forma de trabajo - en LISP, puesto que cuando alguna persona pretende resolver - algún problema utilizando el lenguaje LISP, ésta deberá definir funciones escritas en el lenguaje, orientadas a resolver el problema en cuestión.

Descripción de algunas funciones escritas en LISP.

a) En LISP se utilizan con mucha frecuencia expresiones formadas por funciones CAR y CDR encadenadas.

Por ejemplo:

(CAR (CDR (CAR X)))

cuyo proceso de evaluación sería: obtener el CAR de la lista X, a ese resultado aplicarle la función CDR y a éste último - resultado aplicarle la función CAR, lo cual daría el resultado final de toda la expresión. El método que se ha seguido - para facilitar el trabajo cuando se presentan este tipo de cadenas, es el de definir funciones con nombres adecuados que - ejecutarían las funciones de la cadena.

Por ejemplo para este caso particular se podrá definir la función con nombre CADAR de la siguiente manera:

(CADAR (LAMBDA (L) (CAR (CDR (CAR L)))))

y por lo tanto, si uno desea seguir esta filosofía de trabajo, entonces cualquier otro tipo de cadenas formadas por las fun-

ciones CAR y CDR, darán origen a definir funciones con nombres como (CAAR, CAADR, CDADR, CDDDR, etc.)

b) Nosotros tenemos a disposición la función predicado EQ, la cual compara si sus dos argumentos son átomos --- iguales o no, pero no lleva a cabo ninguna comparación para el caso en que sus argumentos sean listas. Entonces construiremos una función predicado EQUAL cuyo valor será T - si sus argumentos son átomos o listas iguales y tomará el valor F con cualquier otra posibilidad entre sus argumentos.

```
(EQUAL (LAMBDA (X Y) (IF  
    (ATOM X) (ATOM Y) (IF  
        (NULL X) (NULL Y) (IF  
            (EQUAL (CAR X) (CAR Y))  
            (EQUAL (CDR X) (CDR Y)) (QUOTE F) )))  
        )))
```

c) Una función que es muy útil en LISP es la función LIST cuya función es formar una lista con sus argumentos evaluados.

```
(LIST (LAMBDA L L))
```

d) Con anterioridad nosotros hemos descrito la función ULTIMO.

e) La posibilidad de poder invertir una lista queda establecida mediante la función INVIERTE y su función auxiliar

liar INVIERTE*.

```
(INVIERTE (LAMBDA (X) (INVIERTE* X (LIST)) ))  
  
(INVIERTE* (LAMBDA (X Y) (IF  
    (NULL X) Y  
    (INVIERTE* (CDR X) (CONS (CAR X) Y)) ))  
    ))
```

f) En LISP las funciones predicado AND, OR, NOT de la lógica Booleana son extremadamente útiles cuando se quieren construir funciones predicado más complejas.

La construcción de estas funciones y sus funciones auxiliares es la siguiente:

```
(AND (LAMBDA* L* (AND* L*)) )  
  
(AND* (LAMBDA (L4) (IF  
    (NULL L4) (QUOTE T) (IF  
        (EVAL (CAR L4))  
        (AND* (CDR L4)) (QUOTE F) ))  
    ))  
  
(OR (LAMBDA* O* (OR* O*)) )  
  
(OR* (LAMBDA (L4) (IF  
    (NULL L4) (QUOTE F) (IF  
        (EVAL (CAR L4))  
        (QUOTE T) (OR* (CDR L4)) ))  
    ))
```

```
(NOT (LAMBDA (L4) (IF L4 (OR) (AND) ))  
))
```

g) Una de las expresiones de mayor utilidad en ---
LISP es la pseudo-función COND, que toma la siguiente forma:

```
(COND (P1 E1) (P2 E2) . . . (PN EN) )
```

Aquí todos los P_i deben ser funciones predicado -
y los E_i pueden ser funciones de cualquier clase. COND trabaja así:

Se evalúa la función predicado P_1 ; si el resultado de esta evaluación es T, se evalúa E_1 y el valor obtenido que dará como el resultado de toda la expresión COND; si el resultado de P_1 no es T entonces E_1 no es evaluado y se prosigue a examinar el valor de P_2 , y así hasta encontrar algún predicado con valor T. Una vez encontrado el predicado su expresión asociada será evaluada y será el resultado de toda la expresión COND. Si ningún predicado con valor T es encontrado la pseudo-función COND queda indefinida.

```
(COND (LAMBDA* L1 (COND* L1) ))  
  
(COND* (LAMBDA (L2) (IF  
          (EVAL (CAAR L2)) (EVAL (CADAR L2))  
          (COND* (CDR L2) )  
          )))
```

Como un ejemplo de cómo utilizar la pseudo-función COND, miremos una forma diferente de definir la función EQUAL.

descrita anteriormente.

```
(EQUAL (LAMBDA (X Y) (COND
  ( (ATOM X) (EQ X Y) )
  ( (ATOM Y) (QUOTE F) )
  ( (NULL X) (NULL Y) )
  ( (NULL Y) (QUOTE F) )
  ( (EQUAL (CAR X) (CAR Y))
    (EQUAL (CDR X) (CDR Y)) )
  ( (AND) (QUOTE F) ) )
  ))
```

Hasta aquí hemos descrito varias funciones que ilustran la amplísima gama de funciones que se pueden construir. Diremos que toda función programada será una función que pertenecerá a la biblioteca de funciones.

CAPITULO IV

GENERALIDADES

a.IV) Desarrollo Futuro.-

- a) La versión de LISP objeto de este trabajo es --
ágil en la transmisión de datos pero como no se emplea la op--
ción de interrupción de programas (program-interrupt), no re--
sulta todo lo ágil que se puede lograr. Este problema se re--
solverá modificando la rutina de entrada-salida.
- b) Se propone ampliar el repertorio de los aparato--
tos de entrada-salida, agregando al teletipo y lectora de cinta
de papel, lo correspondiente a cintas magnéticas, perfora--
dora de cinta de papel, disco y pantalla catódica (display).
- c) Se propone formar una biblioteca de funciones -
para el sistema.
- d) Se plantea la necesidad de mantener bajo obser--
vación el funcionamiento del interpretador, pues podrán sur--
guir situaciones imprevistas.
- e) Este trabajo puede servir como base para efectuar
reajustes y acondicionamientos del lenguaje LISP a posibles
necesidades futuras, posiblemente para fines más prácticos que
los que permite actualmente el sistema.

b.IV) Aplicaciones.-

En cualquier centro de cálculo es de gran importancia contar con un procesador simbólico como LISP, por dos razones fundamentales.

- 1) Fines pedagógicos.
- 2) Como un lenguaje base para la investigación en teoría de lenguajes, técnicas de programación recursiva y aplicaciones en matemáticas y física, tales como manipulación algebráica, simulación de sistemas y teoría de grupos.

Debemos mencionar aquí la fuerte restricción que - representa el tamaño del computador, pues como se habrá notado, en la definición y operación de las funciones LISP se ocupan gran cantidad de palabras de memoria y a pesar de la - existencia del "colector de basura", el procesador LISP tiene a saturar la memoria del computador, sobre todo cuando no se tiene suficiente experiencia. Debemos aclarar también -- que este hecho no impide que el procesador LISP sea capaz de dar solución a gran diversidad de problemas.

El conocimiento previo de esta restricción hizo -- que la implementación fuera orientada más a fines pedagógicos que a fines prácticos.

Por ejemplo:

- a) La descripción que se hace de manera explícita de los componentes básicos de LISP beneficia el aprendizaje del lenguaje mismo.

b) El hecho de haberse definido un mínimo de funciones y pseudo-funciones como funciones de máquina, hacen -- que el programador se enfrente en forma inmediata al problema de construir funciones más cómodas que faciliten el manejo de sus problemas.

CONCLUSION

Con este procesador de LISP se pueden resolver problemas conectados a las áreas de manipulación algebraica, simulación de sistemas, investigación en teoría de grupos, álgebra moderna, clasificación y manejo de información y otros -- problemas lógicos comunes.

En efecto se han construido funciones que resuelven problemas concretos en estas áreas.

APENDICE A

La función DERIVADA y su función auxiliar POLACA.

Como un ejemplo fácil e ilustrativo para hacer uso de LISP, consideremos el problema de derivar una expresión algebraica.

El problema inicial será el de transformar la expresión algebraica a derivar a una expresión equivalente en forma de lista. Es decir, si la expresión a derivar es $2x^2 + 4$ una expresión equivalente en forma de lista es $((2 * (X * X)) + 4)$. Una vez que la expresión a derivar sea expresada en forma de lista, ésta podrá ser manipulada por funciones LISP.

Una situación que facilitará la construcción de la función DERIVADA será tener la expresión a derivar en notación polaca. Es decir, si la expresión a derivar es $((2 * (X * X)) + 4)$ la expresión equivalente pero en notación polaca será $(+ (* 2 (* X X)) 4)$. (ver ref. 1 T. A. BRODY)

Para efectuar la transformación de una expresión de notación común a notación polaca construiremos la función que llamaremos POLACA.

```
(POLACA (LAMBDA (X) (COND
  ((ATOM X)      (X) )
  ((NULL (CDR X)) (CAR X) )
  ((ATOM (CAR X)) ... )
  (LIST (CADR X) (CAR X) (POLACA (CADDR X)) ) ) )
```

```
( (ATOM (CADDR X))
  (LIST (CADR X) (POLACA (CAR X)) (CADDR X) ) )
( (AND)
  (LIST
    (CADR X) (POLACA (CAR X)) (POLACA (CADDR X))
  )
)
)))
```

Una vez que sabemos como modificar la expresión a derivar a notación polaca, construiremos la función DERIVADA, -- que será una función de dos argumentos.

- 1) La expresión a derivar en notación polaca.
- 2) La variable respecto a la cual se ejecutará la derivación.

DERIVADA es una función cuyo funcionamiento de base en detectar el signo de operación entre dos elementos y actuar en consecuencia, pues reducirá el problema general al problema reducido de derivar una constante o una variable con respecto a la cual se hace la derivación, en cuyo caso el valor será 0 ó 1 respectivamente.

```
(DERIVADA (LAMBDA (E X) (COND
  ( (ATOM E) (IF (EQ E X) (QUOTE 1) (QUOTE 0) ) )
  ( (EQ (CAR E) (QUOTE *)
    (LIST (QUOTE +)
      (LIST (QUOTE *) (DERIVADA (CADR E) X)
        (CADDR E)) (LIST (QUOTE *) (CADR E)))))))
```

- 60 -

```
(DERIVADA (CADDR E) X) ))>
( EQ (CAR E) (QUOTE +)) (LIST (QUOTE +)
(DERIVADA (CADR E) X) (DERIVADA (CADDR E) X) )
( AND) (LIST (QUOTE (CASO IMPREVISTO) ) )
))))
```

Derivar la expresión $2x^2+4$ con estas funciones se haría así:

```
(DERIVADA (POLACA (QUOTE ((2 * (X * X)) + 4) (QUOTE X) )
```

y el resultado arrojado por estas funciones es:

```
(+ (* (+ (* 0 (* X X))(* 2 (+ (* 1 X)(* X 1)))) 0)
```

Hacemos notar que las funciones POLACA y DERIVADA aquí descritas pueden ser expandidas para tratar expresiones más generales que expresiones con sumas y productos. (ver J. Barberan ref. 6). Sin embargo, es necesaria una función que simplifique los resultados arrojados por la función DERIVADA.

APENDICE B

Características del Sistema PDP-15

- a) 24K palabras de memoria.
- b) Palabras de 18 bits.

Registros Especiales

- c) Un acumulador de 18 bits con una extensión de un bit llamado link.
- d) 8 registros auto-incrementables.
- e) Un registro de índice.
- f) Un registro Límite.

APENDICE C

Programa (procesador LISP)

/COMO CARGAR EL PROCESADOR LISP

/EL PROGRAMA LISP SE ENCUENTRA ALMACENADO EN UNA CINTA
/DEC-TAPE, POR TANTO ES NECESARIO PRIMERO CARGAR EN MEMORIA UN
/PROGRAMA LLAMADO DTARUN CUYA FUNCION ES CARGAR EN MEMORIA UN
/UN PROGRAMA ALMACENADO EN UNA DEC-TAPE. ESTE ULTIMO
/PROGRAMA ESTA GRABADO EN UNA CINTA PERFORADA, ASI EL
/PROCEDIMIENTO ES EL SIGUIENTE:
/1) COLOCAR EN LOS APARATOS CORRESPONDIENTES: LA CINTA
/ PERFORADA CON EL PROGRAMA DTARUN
/ Y LA DEC-TAPE DONDE SE ENCUENTRA LISP.
/2) ESTABLECER EN LOS SWITCHS DE LA CONSOLA LA DIR. 01778
/3) CONSECUTIVAMENTE OPRIMIR LAS TECLAS STOP, RESET, READIN,
/ QUE EN SI SON LAS ORDENES PARA EFECTUAR LA LECTURA DE
/ LA CINTA PERFORADA.
/ EN ESTE PUNTO EL CONTROL DEL SISTEMA LO TIENE EL PRO-
/ GRAMA DTARUN, LO CUAL QUEDA DE MANIFIESTO AL IMPRIMIRSE
/ SOBRE EL TTY.
/ DTARUN
/
/4) EN ESTE PUNTO EL SISTEMA ESTA EN ESPERA DEL NOM-
/ BRE DEL PROGRAMA QUE SE QUIERA ESTABLECER EN MEMORIA
/ DESDE UNA DEC-TAPE. ASI, NOSOTROS CONTESTAMOS
/ ILISP
/ A LO CUAL DESPUES QUE SE HAYA EFECTUADO LA LECTURA,
/ EL SISTEMA PDP-15 ESTA LISTO A RECIBIR CUALQUIER PRO-
/ PROGRAMA LISP.

.END START

KES#7...332

RRA#7VV322
•TITLE LISP
•ABS
•LOC 10

13777 /APUNTADOR DE PUSH (VALOR INICIAL)
0 /ES USADA POR POP2
23777 /APUNTADOR DE PUSH1 (VALOR INICIAL)
37777 /APUNTADOR DE PUSHA (VALOR INICIAL)
0
40777 /APUNTADOR DE PUSH0.

.LOC 5b

/REGION DE LETREROS

	031714 /COLECTOR
	050324
	172215
	040500 /LE
	020123 /MASURA
	252211
	1
LISP	141123
	205562
	641300
IREG	115515
	556555
	164100
SPROS	520522
	221722
	520041
PC	200375
	0
AC	010375 /AC# ES DE LETRERO
	0

.LOC 100

/FUNCIONES PRIMITIVAS (LOC. 100-137)

CARO

030122

0

0

CDRO

030422

0

0

CONS

31716

230000

0

ATOH

12417

150000

0

EQ

052100

0

0

PRINT /SERÁ TRATADA COMO UNA FUNCION

202211

162400

0

EVALL /EVAL

052601

140000

0

/LOC VACIA PARA OTRA FUNCION

0

0

0

LOC 148
/FORMAS LISP (LOC 148-217)

QUOTE
212517
240590
0
DEFINE
040506
111605
0
NULL NULL
162514
140000
0
DDAA LAMBDA /LAMBDA#
140115
\$29481
\$29000
DDA LAMBDA
140115
\$29481
0
IFF IF /IFF ES ETIQUETA DE REFERENCIA.
110600
0
0
V 0 /APUNTADOR DE V
260000
0
0
F 0
060000
0
0
ENTR /ENTRADA
051424
226104
010600

/LOC VACIAS PARA OTRAS FORMAS

.LOC 424

/RUTINA READ
/INTERPRETA LA NOTACION LIST

READ	0		
	JMS	COINP	/CONDICIONES INICIALES PARA EMPAQUETAR ATOMOS.
OTROBU	JMS	UNORU	
	JMS	COINBU	/CONDICIONES INICIALES DE BUFFER
	JMS	BUFFE	/SE LLENA EL BUFFER
	JMS	COINBU	/COND. INICIALES DE BUFFER
	CLAICLL		/LINK# INDICA QUE NO HAY PAQUETE DE ATOMOS.
OTCHA	JPS	DEBUF	/SE DESALOJA EL BUFFER CHA POR CHA.
	JMP	OTROBU	
	SAD	C240	
	JMP	SPACE	/A PROCESAR UN ESPACIO
	SAD	C250	
	JMP	PARIZO	/A PROCESAR UN PARENTESIS '('
	SAD	C251	
	JNP	PARDER	/A PROCESAR UN PARENTESIS ')'.
	JMS	EIPAS	/PROCESA UN CHA PARA ATOMO PONE LINK#1
	JNP	OTCHA	/A PROC. EL SIG. CHA. DE EL BUFFER.
	SZL		(PRUEBA) 1 HAY PAQUETE 0 NO HAY PAQUETE.
	JMP	OTCHA	/NO HUBO PAQUETE.
	CLAICEL		/LINK# ES BANDERA DE ESPACIO.
	SAD	BPIZO	/ATOMO DENTRO DE UNA ESTRUCTURA O NO.
	JMP	SOLO	/A PROCESAR UN ATOMO FUERA DE UNA ESTRUCTURA.
	JMS	PRUATO	/BUSCA ENCUENTRA Y PROCESA ATOMOS
	SZL		(PRUEBA) 0 FUE ESPACIO 1 NO FUE ESPACIO.
	JNP	OTCHA	
	CLL		
SPRIG	JMS	PARD	/PROCESA UN ')' DENTRO LA ESTRUCTURA DE LISTA.
	JNP	OTCHA	/LA ESTRUCTURA AUN NO ESTA COMPLETA.
	LAC	NOQINI	
	JNP*	READ	/ACERRODO INICIAL DE LA LISTA LEIDA.
PARDER	SZL		
	JNP	PARIGH	(PRUEBA) BANDERA DE PAQUETE.
	JNP	SPRIG	/SE PROCESA UN ')'.
PARIZO	SZL		
	JIS	PRUATO	/LINK#1 HAY PAQ. DE ATOMO
	CLL		/BUSCA ENCUENTRA Y PROCESA ATOMOS
	JIS	PARIZ	/HORA BAND. DE PAQUETE SI DA HAY.
	JNP	OTCHA	/PROCESA UN '('.
SOLO	JIS	SEARCH	
	JNP	OTCHA	/SOLICITANTE PROCESA UN ATOMO SOLO.

/RUTINA PARIZ
/PROCESA LOS PARIZ-TESTS (IZQUIERDO).

PARIZ	#		
LAC	INTEN	/ (PRUEBA) BAND. DE PAR. IZQUIERDO.	
SNA			
JMP	PRIME	/ES EL PRIMER PARIZ-TESTS. IZQUIERDO.	
LAC	NODCO	/AL AC EL NODO CORRIENTE.	
JRS	CARRO	/SE OBTIENE EL CAR DEL NODO DIR. POR EL AC.	
SNA		/ (PRUEBA) ATOMO-LISTA O NADA(CERO).	
JHP	CERO		
LAC	NODCO	/FUE ATOMO-LISTA.	
JRS	CORRO	/AL AC EL CAR DEL NODO DIR. POR EL AC.	
DAC	NODCO	/PONE A DISPOSICION UN NUEVO NODO.	
ARRIBA	JHS	PUSH	/PONE ESE NODO EN EL PUSH.
JHS	CORRO	/SACA EL CAR DEL NODO CORRIENTE.	
DAC*	NODCO	/PONE EL AC EN EL CAR DEL NODO CORRIENTE.	
DAC	NCOCO	/PONE A DISPOSICION EL FUEG DIR. POR EL COR.	
JHP*	PARIZ		
CERO	LAC	NODCO	
JHP	ARRIMA		
PRIME	ISZ	BPIZO	/PONE DIF. DE CERO LA BAND. DE PAR. IZO.
JHS		MEMOR	/OBTIENE EL PRIMER NODO LIBRE.
DAC	NODCO		/SE PONE A DISPOSICION EL PRIM. NODO VACIO.
DAC	NODINI		/SE DEP. EL PRIM. NODO USADO.
JHP*	PARIZ		

/LOC ESPECIALES QUE USA PARIZ.

BPIZO		
PRIVA	24004	/MAND. DE PAR. IZO. (0 AL INICIO)
NODCO		/PRIMER NODO LIBRE (SIEMPRE).
NODINI	0	/PRIMER NODO CORRIENTE.
		/VALOR DE READ A LA SALIDA.

/RUTINA PROATO
/BUSCA UN ATOMO Y PROCESA
/ESTE DENTRO DE LA ESTRUCTURA DE LISTA.

PROATO	#	
JHS	SERCH	/BUSCA UN ATOMO (IR APUNTA A ESTE)
JHS	REATO	/PROC. EL ATOMO DENTRO LA ESTRUCTURA DE LISTA
JHP*	PROATO	

/RUTINA TECNA
 /RUTINA QUE CUELA AL RT, LF, RUBOUT DADOLE AL RT, LE CARAC-
 /TERISTICAS DE TERMINADORES DE BUFFER, AL RUBOUT -BURNADOR
 /DE CARACTERES EN EL BUFFER
 /Y A LOS DEMAS SIGNOS NO LES HACE NADA

TECNA	S		
SAD	C212		
JMP	LF	/LA COLADERA PROCESA LINE-FEED.	
SAD	C215		
JMP	RETLF	/LA COLADERA PROCESA UN RETURN.	
SAD	C377		
JMP	RUBOUT	/LA COLADERA PROCESA UN RUBOUT.	
SAD	CCTRU		
JMP	CTRLU	/LA COLADERA PROCESA UN CONTROL U.	
CLE		/BANDERA QUE INDICA QUE CHA PASO LA COLADERA.	
JMP*	TECNA		
RETLF	JMS	P77	
LF	LAC	C248	/SIMULARA UN ESPACIO
	DAC	APUNT	/COMPLETA LA SIMULACION.
	ISE	APUNT	/METIO UN ESPACIO AL BUFFER.
	B2H	APUNT	/DELIMITADOR DE EL BUFFER
	LAC	C212	
MOP1	JMS	P77	/RUTINA QUE ESCRIBE UN CHA POR EL TTY.
	STL		/BAND. QUE IND. QUE CHA SE QUEDO EN LA COLADERA.
	JMP*	TECNA	
RUBOUT	LAC	BRUBO	/BANDERA DE RUBOUT
	BRB		/PREG. SI ES EL PRIM. RUBOUT
	JMP	+4	
	ISE	BRUBO	/PONE BANDERA DE RUBOUT.
	LAC	C334	/EN CODIGO ASCII DE DIAGONAL.
	JMS	P77	
	LAC	CONT	/CHECO EL VALOR DEL CONTADOR
	SAD	(777670)	/=-72 DECIMAL
	JMP	DEC72	
TAD	(-1)		/RED. EN 1 EL CONTADOR
DAC	CONT		/NUEVO VAL DEL CONT.
CLC			/AC=-1
TAD	APUNT		/RED. EL VAL DE APUNT.
DAC	APUNT		/NUEVO VAL DEL APUNTADOR.
LAC	APUNT		
JMP	LF+8		
CTRLU	LAW	336	/AL AC EL COD. ASCII DE "
	JMS	P77	
	LAW	328	/AL AC EL CODIGO ASCII DE U
	JMS	P77	
	BRB	COINGU	/CONDICIONES INICIALES DE BUFFER.
DEC72	JMS	RTLF	/CONT=-72 DECIMAL.
	B2H	BRUBO	/BORRO LA BANDERA DE RUBOUT.
	JMP	LF+6	
BRUBO	S		/BANDERA DE RUBOUT.

/RUTINA BUFFER
/RUTINA QUE FORMA UN BUFFER DE A LO MAS 72 CARACTERES.
/PERO PUEDE SER CERRADO ANTES CON UN BT O LF. P. EJEMPLO:
/COMO DELIMITADOR UN CERO AL FINAL.
/CONT=7776700-72(DECIMAL AL PRINCIPIO).
/APUNT=4000 (ESTE ES LA DIR. INICIAL DEL BUFFER).

BUFFER	S	LETTY	/RUTINA QUE LEER UN CHA POR EL TTY.
LEERCHA	JMS	TECHA	
	JMS		
	CLA		
	SAC*	APUNT	
	JNPD	BUFFER	/BUFFER CERRADO
	SIL		
	JNP	.-6	/A NUEVO CHA PUES SE LEYO UN RUBOUT,LF,RT.
	LAC	BRUG	/PRIMERA SI ESTA PUESTA LA BANDERA DE RUBOUT.
	SZA		/0 SIGNIFICA QUE NO ESTA PUESTA.
	JNP	FINRUD	/ESTA PUESTA LA BANDERA DE RUBOUT.
CHAR	LAC	CMA	/ULTIMO CHA LEIDO
	DAC*	APUNT	
HOP2	JMS	PTTY	/RUTINA QUE ESCRIBE UN CHA POR EL TTY.
	ISL	APUNT	
	ISL	CONT	
	JNP	BUFFER+1	/A VOLVER A LEER UN CHA
	DZM*	APUNT	/SE CIERRA EL BUFFER
	JNPD	BUFFER	/BUFFER LLENO
FINRUD	LAC	C334	/CODIGO ASCII DE DIAGONAL.
	JMS	PTTY	
	DZM	BRUG	/LIMPIA LA BANDERA DE RUBOUT.
	JNP	CHAR	

/RUTINA DEBUF
 /RUTINA QUE DESALOJA UN BUFFER DE A LO MAS 72 CHA
 / O HASTA QUE ENCUENTRE UN DELIMITADOR CERO.
 /EMPIEZA A DESALOJAR CHA A PARTIR DE LA DIRECCION
 / MARCADA POR EL APUNTADOR COL. EN LA LOC. APUNT.
 /ESTA RUTINA TIENE DOS SALIDAS.
 / SALIDA 2 - SALE CON UN CHA EN EL AC.
 / SALIDA 1 - INDICACION DE QUE SE ACABA EL BUFFER.

DEBUF	0
LAC0	APUNT
SAD	(S)
JMP	ACABO
ISE	APUNT
ISE	DEBUF
JMP0	DEBUF

ACABO	
-------	--

/RUTINA PARDE
 /PROCESA EL PARENTESIS DERECHO (READ)
 /TIENE DOS SALIDAS
 / 1) PROSIGUE LA ESTRUCTURA Y EL PROC. DE LECTURA.
 / 2) FIN DE LA ESTRUCTURA Y MARCA EL REGRESO DE (READ)

PARDE	0	
LAC	NODOC	
DAC	TEMP2	/LO CUAL NOS PERMITE TENER ACCESO AL CDR.
JMP	CERO	/AL AC EL CDR DEL NODO CORRIENTE.
DAC	TEMP3	/SE ALMACENA TEMPORALMENTE.
CLA		
JMP	DEPCDR	/DEP. CERO EN EL CDR DEL NODO CORRIENTE.
LAC	I2	
SAD	ZIPOL	/INMEDIATAMENTE MIRA SI ZIP ES VACIO.
JMP	ENIR	
JMP	POP	/ZIP NO ES VACIO.
DAC	NODOC	
DAC	TEMP1	
LAC	TEMP4	
JMP	DEPCDR	
JMP0	PARDE	/SALIDA 1
ISE	PARDE	
SAD	DEI2Q	/LIMPIA LA SUND. DE PAR. I2Q. (PRIMERO)
LAC	TEMP4	
DAC	PRIVA	/DEP.DIR. PRIM. LOC. LIBRE
JMP0	PARDE	/SALIDA 2

/RUTINA REATO

REATO	#	LAC	NODCO	COMENTARIO
	8	JHS	CARNO	/AL AC EL NODO CORRIENTE.
		SNA		/AL AC EL CAR DEL NODO CORRIENTE.
		JNP	NADA	/(PRUEBA) ATOMO-LISTA 0 NADA
		LAC	NODCO	/FUE CERO EL CAR
		JHS	CORNO	/AL AC EL CAR DEL NODO CORRIENTE.
		DAC	NODCO	/SE COLOCA OTRO NODO CORRIENTE.
NADA		PXA		/IR AL AC
		DAC*	NODCO	/AC AL CAR DEL NODO CORRIENTE.
		JMP*	REATO	

/RUTINA UNOBUS
/MARCA LAS LOC. QUE USARA EL BUFFER PARA
/DETECTOR EL DELIMITADOR 0.

UNOBUS	#	LAC	(777676)	COMENTARIO
		DAC	CONT	
		LAC	(3777)	
		DAC	17	
		DAC*	17	
		ISZ	CONT	
		JNP	-2	
		JMP*	UNOBUS	

/RUTINA CARNO
/OBTIENE EL CAR DE UN NODO (EL NODO QUE SE QUIERE VER SE
/PONE EN EL AC AL PRINCIPIO DE LA RUTINA.

CARNO \$
DAC TEMP2
LAC# TEMP2
JMP# CARNO

/RUTINA CDRNO
/OBTIENE EL CDR DEL NODO (EL NODO QUE SE
/QUIERE ABRIR SE PONE EN EL AC).

CDRNO \$
DAC 17
LAC# 17
JMP# CDRNO

/RUTINA DEPCDR
/METE EL CONTENIDO DE EL AC AL CDR DEL NODO DIREC-
/CIONADO POR TEMP2.

DEPCDR \$
DAC TEMP3
LAC TEMP2
DAC 17
LAC TEMP3
DAC# 17
JMP# DEPCDR

/LOC ESPECIALES UTILIZADAS POR ESTAS RUTINAS.

TEMP2 \$
TEMP3 \$
TEMP4 \$

/LOC, USADA POR LA RUTINA PARDE.

/RUTINA CLTRES
/LIMPIA TRES PALABRAS GMPEZANDO CON LA QUE ES APUNTADA
/POR EL AC Y REGRESA CON EL MISMO VALOR EN EL AC.

CLTRES 0
TAD (-1)
DAC 17
D2H# 17
D1H# 17
D2H# 17
IAC
JMP* CLTRES

/RUTINA COPUSH
/PONE COND. INIC. A PUSH2 Y A PUSH1

COPUSH 0
LAC (23777) /APUNT. DE INICIO DE PUSH1
DAC 12
LAC LIPDL /APUNT. DE INICIO DE PUSH2
DAC 10
LAC (-1) /CONST. DE PDL DOBLE VACIO
DAC CAR
DAC CDR
JMP* COPUSH

/RUTINA VACONS
/PONE VALORES INIC. EN LOC. ESPECIALES.

VACONS 0
LAC (220)
PAL /220 AL LR (PARA TENER LAS FUNC. DEL SISTEMA)
D2H BPIZQ /LIMPIA BANDERA DE PAR. IZQ.

LAC (24001)
DAC 12
LAC (24000) /AL AC EL APUNTADOR DE VACIO.
DAC# 12 /24002=24000
D2H# 12

LAC (24004)
DAC PRIVA /INIC. DE LA PRIM. LOC. DE CADENA.
JMS COPUSH /COND. INIC. A PUSHES
JMP* VACONS

/RUTINA COINBU
/ESTABLECE LAS CONDICIONES INICIALES PARA USO DEL BUFFER.
/PRI= DIRECCION INICIAL DEL BUFFER.

COINBU	#	
LAC	(4000)	
DAC	APUNT	/APUNTADOR DEL BUFFER
LAC	(777670)	/ESTO ES UN -72 DECIMAL.
DAC	CONT	/CONTADOR DEL BUFFER.
JMP*	COINBU	
APUNT	#	/APUNTADOR DEL BUFFER (INICIALMENTE 4000)
CONT	#	/CONTADOR DEL BUFFER (INICIALMENTE 777670=-72)

/RUTINA ENLACE
/RUTINA QUE ENLAZA LA MEM. PARA FORMAR LA REGION DE LISTAS
/PONE CERO COMO VALOR DE TODOS LOS ATOMOS
/(ENLAZA LAS DIRECCIONES 24000 HASTA 37777 TEMPORALMENTE,

ENLACE	#	
LAC	(4000)	/LIMITES SUP. SE ATOMOS (ABIERTO)
PAL		/AC AL LR
LAC	(220)	/LIMITES INFERIOR DE ATOMOS (CERRADO)
PAX		/AC AL IR
DZM	0,X	
AXS	4	/SUMA 4 A IR Y SALTA SI IR>LR
JMP	-2	/ITERA
LAC	(23777)	
DAC	17	
CLA		/SON LOS LUGARES PARA EL VACIO.
DAC*	17	
DAC*	17	

OTRO	CLA	/SE ENLAZAN LAS LOCALIDADES.
DAC*	17	
LAC	17	
AAC	2	
DAC*	17	
SAD	(40000)	/ESTO ES UN NUMERO TFM. PARA USAR DDT.
JMP*	ENLACE	
JMP	OTRO	

/RUTINAS PUSH Y POP (DE LA MANERA MAS SIMPLE).
/INICIALMENTE 10 DEBE TENER 13777 COMO SU CONTENIDO

PUSH	0	/PUSH METE EL CONTENIDO DEL AC AL PDL.
	DAC	TEMP
	LAC	10
	SAD	LSPDL / (PRUEBA) LIMITE SUP. DE EL PDL.
	JMS	ERROR / (ESTA LLENO EL PDL).
	LAC	TEMP
	DAC*	10
	JMP*	PUSH
POP	0	/PONE EN EL AC EL PRIMER REGISTRO DISP., EN EL PDL.
	LAC	10
	DAC	TEMP
	TAD	(-1) /ESTO ALTERA EL LINK ' LO COMPLEMENTA'.
	DAC	10
	CHL	/ESTO REGRESA SU VALOR AL LINK.
	LAC*	TEMP
	JMP*	POP

/TRES LOC ESPECIALES PARA ESTAS RUTINAS.

LIPDL	13777	/LIMITE INFERIOR DE EL PDL
LSPDL	23777	/LIMITE SUPERIOR PARA EL PDL.
TEMP	0	

/RUTINA PUSH A POP.
/SON USADAS UNICAMENTE POR LAMBDA AL ENLAZAR.

PUSHA	0	
	DAC*	13
	JMP*	PUSHA
POPA	0	
	LAC	13
	DAC	TEMP
	TAD	(-1)
	DAC	13
	CHL	/COMPLEMENTA EL LINK
	LAC*	TEMP
	JMP*	POPA

/RUTINA COMPA
/COMPARA TRES PALABRAS DE UNA AREA FIJA
/A PARTIR INCLUSIVO DE LA DIRECCION 'SIXPTR+1')
/CON OTRA TRES PALABRAS LAS CUALES
/SON DIRECCIONADAS POR EL IR.
/LA RUTINA TIENE DOS SALIDAS
/SALIDA 1 -----FALSO
/SALIDA 2 -----VERDAD

CONPA 8

LAC: (SIXPTR)
DAC: 17
LAC*: 17
SAD: 1,X
JMP: UNA
JMP*: CONPA /SALIDA 1
UNA
LAC*: 17
SAD: 2,X
JMP: DOS
JMP*: CONPA /SALIDA 1
DOS
LAC*: 17
SAD: 3,X
JMP: CONPA /SALIDA 2
JMP*: CONPA /SALIDA 2

/RUTINA COINP
/ESTABLECE LAS COND. INIC. PARA EL EMP. DE CHA.

COINP 9

DEM: BYTPTR /SE PONE A CERO EL APUNTADOR DE BYTE.
LAC: (SIXPTR+1) /AL AC LA PRIM. PAL. PARA EMP. ATOMOS.
DAC: SIXPTR
JNS: CLTRES /LIMPIA TRES PALABRAS.
LAN: 17766 /CARGO AC CON UN -10 DECIMAL.
DAC: CONTO /CONTADOR DE PAQUETE DE CHA.
JMP: COINP

CONTO 9

BYTPTR 6
SIXPTR 9

NOP
NOP
NOP
/CONT. DE CHA PARA EL PAQUETE (IN. -10)
/APUNT. DE BYTE EN UNA PAL. DE EL PAQ.
/APUNT. DE PAL. DEL PAQ. (IN. SIXPTR+1)
/PRIM. PAL. PARA EL PAQUETE.
/SEG. PAL. PARA EL PAQ.
/TERC. PAL. PARA EL PAQ.

/RUTINA SEARCH
/BUSCA UN ATOMO SOBRE LA REGION DE ATOMOS.
/OPCION 2 -- SE LLENO LA REG. DE ATOMOS SE DETIENE EL PROG.
/OPCION 1 -- SE BUSCA (LO ENCUENTRA
/O NO LO ENCUENTRA PERO ENTONCES LO PONE).
/REGRESA CON IR APUNT. A SU LUGAR Y CON SU VALOR EN EL AC.
/NOTA EN COND. INICIALES SE PONE LR=100 (OCTAL).

	SERCH	0	
	LAC	(74)	
	PAX		/AC AL IR
LOOP	AXS	4	/SUMA 4 A IR Y SALTA SI IR>LR.
	JMP	BUSCA	
	PLA		/COLOCA LR AL AC
	TAD	(774000)	/(PRUEBA) SI SE LLENO LA REG. DE ATOMOS.
	SMAICLA		
	JHS	ERROR	/OPCION 2 SE LLENO LA REGION DE ATOMOS.
	PLA		/LR AL AC
	TAD	(4)	
	PAL		/AC AL LR
	JNS	PONE	
	JMP	SALE1	
BUSCA	JMS	COMP1	
	JMP	LOOP	/NO FUE IGUAL (SALIDA FALSA).
	JNS	COINP	/SE PONEN LAS CONDICIONES INICIALES DE PAQUETES.
	CML		/PRESERA CONSTANTE EL VAL. DEL LINK.
	LAC	#,X	/SE ENCONTRO O SE PUSO EL ATOMO EN PROCESO.
	JMP*	SEARCH	

/RUTINA PONE
/COLOCA EL CONTENIDO DE TRES PALABRAS DE UN AREA FIJA
/(LA PRIMER LOCALIDAD DE LAS CUALES ES SIXPTR+1)
/SOBRE OTRAS TRES PALABRAS DIRECCIONADAS POR EL IR.

	PONE	0	
	LAC	(SIXPTR)	
	DAC	17	
	LAC*	17	
	DAC	1,X	
	LAC*	17	
	DAC	2,X	
	LAC*	17	
	DAC	3,X	
	JMP*	PONE	

/RUTINA SIXBT
/EMPAQUEA TRES CODIGOS ASCII EN UNA PALABRA Y SE
/PUEDEN ALMACENAR TANTOS CODIGOS COMO SE QUIERAN
/(EL COD. A EMP. SE COLOCA EN EL AC AL LLAMAR LA RUTINA).

SIXBT	0	
	AND	(77)
	DAC	TEMP
	LAC	BYTPTR
	TAD	(JMP# TABYTE)
	DAC	.+3
	ISZ	BYTPTR
	LAC	TEMP
	MOP	
TABYTE	BYTE1	
		BYTE2
		BYTE3
BYTE1	SWHA	
	CLLIRAL	
	RTL	
	XOR#	SIXPTR
	DAC#	SIXPTR
	SEL	
	ISZ	SIXPTR /SE INCREMENTA PARA LLENAR OTRA PALABRA.
	JMP#	SIXBT
BYTE2	CLLIRTL	
	RTL	
	JMP	BYTE1+2
BYTE3	STL	/LINK#1 ES BANDERA DE BYTE3
	DEN	BYTPTR
	JMP	BYTE1+3

/RUTINA ENPA9
/EMPAQUEA A LO MAS 9 CARACTERES
/EL COD. ASCII DEBE IR EN EL AC CADA VEZ QUE SE LLAME
/LA RUTINA. EL CONTADOR DE CARACTERES Y EL APUNTADOR
/DE LA PALABRA USADA PARA EMPAQUETAR DEBERA HABER
/SIDO ESTABLECIDA CON ANTERIORIDAD.

ENPA9	0	
	ISZ	CONT0 /COMO COND. INIC. DEBE TENER UN -10 DECIMAL
	SKP	
	SKP	/ATOMO DE MAS DE 10 CARACTERES.
	JMS	SIXBT /A EMPAQUETAR EL CMA
	STL	/BANDERA DE EMPAQUETAMIENTO DE ATOMOS.
	JMP#	ENPA9
/LOC. ESPECIALES DE ESTA RUTINA.		
/RUTINA LEETTY		
/LEE UN CARACTER DEL TTY.		

LEETTY	0	
	KSF	
	JMP	.-1
	KRS	

CHA

ADS
AAC
DAC
JNPs
LEETTY
CHA

ADS
AAC
DAC
JNPs
TCE
TSP
TLP
PRTY

RSA
HSP
JMP
HRH
AND
JNPs
GND
CHB
DAB
JNPs
LEETTR

/RUTINA LEETTR

/RUTINA QUE IMPRIME UN CMA QUE SE ENCUENTRA EN EL AC.

(177) /ES UN NUMERO OCTAL.

HELR

ADS
AAC
DAC
JNPs
TCE
TSP
TLP
PRTY
C213
LAC
JHS
DTTY
C212
HTTY
JNPs
HELR

/RUTINA MTRP
/DESCRIBE UN RETURN LINE-PRED.

/VALORES CONSTANTES QUE UTILIZA EL PROGRAMA

C212	212	/LINE FEED	(CODIGO ASCII)
C215	215	/RETURN	(CODIGO ASCII)
C269	269	/CEHO	(CODIGO ASCII)
C248	248	/ESPACIO	(CODIGO ASCII)
C250	250	/PARENTESIS IZQ.	(CODIGO ASCII)
C251	251	/PARENTESIS DERECH.	(CODIGO ASCII)
CCTRLU	225	/CTRL U	(CODIGO ASCII)
C377	377	/RUBOUT	(CODIGO ASCII)
C334	334	/DIAG. INV.	(CODIGO ASCII)

/RUTINA PATOM
/ESCRIBE UN ATOMO EL CUAL ES APUNTADO POR
/EL IR Y CUANDO CONCLUYE SE ESCRIBE UN ESPACIO.

PATOM	0	
	LAM	-3
	DAC	CONT
	DEM	BYTPTR /LIMPIA EL APUNTADOR DE BYTE,
	DZM	SIXPTR /LIMPIA EL APUNTADOR DE PALABRA,
OTOCHA	LAC	SIXPTR
	TAD	SIXPTR
	DAC	(JMP# TABYTA)
	NOP	+1
TABYTA	BYTA1	
	BYTA2	
	BYTA3	
BYTA1	LAC	1,X
	JMP	UNI
BYTA2	LAC	2,X
	SKP	/JMP +2
BYTA3	LAC	3,X
	UNI	DAC TEMP
		LAC BYTPTR
		TAD (JMP# TABYTO)
	DAC	+3
	ISZ	BYTPTR /INCREMENTA EL APUNT. DE BYTE.
	LAC	TEMP
	NOP	
TABYTO	BYTO1	
	BYTO2	
	BYTO3	
BYTO1	SWHA	/AQUI SE ESCRIBIRA EL PRIMER BYTE SEGUN PALPTR.
	RAR	
	RTR	
MASC	AND	(77)
	SZA	
ESPACE	JMP	INPRI /A IMPRIMIR UN CHA Y REGRESAR POR OTRO SU HAY.
	LAC	(40) /SE FORMARA EL CODIGO DEL ESPACIO.
	JMP	--2 /LO CUAL QUIERE DECIR QUE SE ACABO EL ATOMO.
BYTO2	RTR	/AQUI SE ESCRIBIRA EL SEG. BYTE SEG. PALPTR.
	RTR	
BYTO3	JMP	BYTO1+2 /REINICIALIZA EL APUNTADOR DE BYTE.
	DZM	BYTPTR /SE INCREMENTA EL CONT. DE PALABRA.
	ISZ	SIXPTR /SIRVE PARA INPR. NO MAS DE 9 CHA.
	ISZ	CONT
	JMP	MASC
INPRI	JMS	MASC /COMPLETA EL CODIGO DEL CHA ASCII.
	JMS	PITY
	ISZ	CONTCA /ES EL CONTADOR DE LINEAS DE SALIDA.
	SAD	C240 /PREG. SI EL ULTIMO CHA ESCRITO FUE UN ESPACIO.
	JMP#	PATOM /SE TERMINO DE ESCRIBIR EL ATOMO.
	CLA	
	SAU	CONT /PREG. SI YA SE ESCRIBIERON 9 CHA.
	JHP	ESPACE
	JHP	OTOCHA

/RUTINA COMPLE
/RUTINA QUE COMPLETA EL CODIGO ASCII

COMPLE 0
TAD (-40)
SPA
JMP TAD340
TAD C240
JMP# COMPLE
TAD340 TAD (340)
JMP# COMPLE

/RUTINA RINS.
/INICIALIZA LOS APUNTADORES EN LA BUSQUEDA DE UNA LISTA.
/PENSAREMOS QUE EL AC CONTIENE EL DATO NECESARIO PARA
/PODER RECONOCER DE QUE LISTA SE TRATA.

RINS 0
DAC S
IAC 10
DAC P /APUNTADOR DEL PDL.
JMS LINEA /LINEA NUEVA E INICIO DE CONTADOR.
DZH BAND /BAND DE INICIO DE IMPRESION A CERO.
JMP# RINS

/LOC. ESPECIALES.

P 0 /APUNTADOR DEL PDL.
S 0 /APUNTADOR DEL SIGUIENTE ELEMENTO.
L 0 /MITAD IZQUIERDA (CAR).
R 0 /MITAD DERECHA (CDR).
RIT 0 /LOCALIDAD TEMPORAL.
BAND 0 /BANDERA DE INICIO EN LA SALIDA DE UNA LISTA
SIMPRI 0 /BANDERA DE IMPRESION.
CONTCA 0 /CONT. DE CAR EN LA LINEA DE SALIDA.

/RUTINA PRESPA
/PRESPA PREG. POR ESPACIOS DISP. EN LA LINEA DE SALIDA.

PRESPA 0 /EL NUM. DE ESP. QUE QUIERE LOS TRAE EN EL AC.
TAD CONTCA
GMA
JMS LINEA /LINEA NUEVA E INICIO DE CONTADORES.
JMP# PRESPA

/ RUTINA LINEA
/LINEA DA UNA LINEA NUEVA E REINICIA LOS CONTADORES.

LINEA 0
JMS RTIE
LAW 177670 /-72 DECIMAL
DAC CONTCA
JMP# LINEA

/RUTINA SIG
/SIG SERA USADA POR LA RUTINA PLISTA Y EL COLECTOR DE BASURA
/MARCARA EL FLUJO CUANDO SE SIGUE UNA LISTA POR LA MEMORIA.

SIG	0	
LAC	S	
SNA		
JMP	END0	
DAC	RIT	
LAC*	RIT	
AND	(77777) /QUITA BAND. DE COLECTOR	
DAC	L /SE OBTIENE EL CAR DE EL NODO.	
ISZ	RIT	
LAC*	RIT	
DAC	R /SE OBTIENE EL CDR DE EL NODO S	
LAC	L	
TAD	(-24880)	
SNA!CLA		/PRUEBA PARA MIRAR SI ES LISTA LA PARTE IEG.
JMP	LISTA	/RESULTO SER LISTA.
LAC	R	/NO RESULTO SER LISTA.
DAC	S	
SZA!CLA		
JMP	ENDEST+1	
END0	LAC (2)	/BANDERA DE FIN DE LISTA.
	DAC BIMPRI	/BANDERA DE IMPRESION
	LAC 10	
	SAD P	/PRUEBA SI PDL ES VACIO.
	JMP ENDEST	/ESTA VACIO EL PDL.
	JRS POP	
	DAC S	
	JMP*	SIG
ENDEST	LAC (3)	/BANDERA DE FIN DE ESTRUCTURA.
	DAC BIMPRI	/BANDERA DE IMPRESION.
LISTA	JMP*	SIG
	LAC L	
	DAC S	
	LAC R	
	JRS PUSH	
CLAS!AC		/PONGO UN UNO EN EL AC PARA BIMPRI
JNP	ENDEST+1	

/RUTINA PRINCH
/IMPRIME ATOMOS, PAF. DER., PAR. 120.
/SEGUN LO INDIQUE LA RUTINA SIG.

PRINCH	8	LAC	BAND	/ORIGINAL ALFETO CERO. SNA
		JMP	BANDO	/PRUEBA A LA BANDERA (V=0,1)
TESTL		LAC	L	
		TAD	(=100)	
		SPA		
		JMP	BPRI	/NO ES ATOMO L.
		LAC	L	
		TAD	(=4000)	
		SNA		
		JMP	BPRI	/NO ES ATOMO L.
		LAC	L	/RESULTO SER ATOMO L.
		DZM	L	
		PAX		
		LAC	(12)	/AC) AL IR
		JNS	PRESPA	/PREG. SI CAE UN ATOMO EN LA LINEA DE IMPRESION.
		JMS	PATOM	/A IMPRIMIR EL ATOMO INDICADO POR L.
		LAC	BIMPRI	/SE EMPEZARA A PROBAR LA BANDERA DE IMPRESION.
		SNA		
		JMP*	PRINCH	
APRI		LAC	(2)	/PRUEBA DE ESPACIO EN LA LINEA DE SALIDA.
		JMS	PRESPA	
		ISZ	CONTCA	/SE INCREMENTA EL CONTADOR DE LINEA POR 2.
		ISZ	CONTCA	
		LAC	BIMPRI	/SE PRUEBA LA BANDERA DE IMPRESION.
		SAD	(1)	
		JHP	D	
		SAD	(2)	
		JMP	O	
		SAD	(3)	
		JHP	F0	
		JMS	ERROR	/SE DETIENE POR UN ERROR.
		LAW	250	/CARGO EL COD. ASCII (,
		JMS	PTTY	
		LAC	C240	/CARGO EL COD. ASCII DEL ESPACIO.
		JMS	PTTY	
		JMP*	PRINCH	
0		LAW	251	/CARGO EL COD. ASCII DE).
		JHP	D+1	
+0		DZM	BAND	
		ISZ	PRINCH	
		JHP	O	
		BANDO		
		ISZ	BAND	
		LAW	250	
		JMS	PTTY	
		LAC	C240	
		JMS	PTTY	
		JHP	TESTL	

/RUTINA PLISTA
/PLISTA ESCRIBE SOBRE EL TTY UNA LISTA QUE ESTE
/GUARDADA EN LA MEMORIA.
/EL AC CONTIENE EL NUMERO DONDE EMPIEZA LA LISTA.

PLISTA	0
JMS	RINS
JHS	SIG
JHS	PRINCH
JHP	.-2
JHP*	PLISTA

/RUTINA PLISP
/PLISP ES LA RUTINA CAPAZ DE IMPRIMIR UN ATOMO
/O UNA LISTA SEGUN SEA EL CASO DEBIDO AL VALOR DEL
/AC (APUNTE A UN ATOMO O A UNA LISTA).

PLISP	0	
DAC	AUXP	/DEP. EN UNA LOC. AUXILIAR
SNA		/EVITO QUE VALGA CERO EL AC.
JMS	ERROR	
JHS	TESTAC	/ANALIZO EL VAL. DEL AC.
JMP	ATOM	/EL AC APUNTA A UN ATOMO.
JHS	PLISTA	/EL AC APUNTA A UNA LISTA.
JHP	IGUAL	/RUMBO A LA SALIDA DE PLISP
ATOM	RTLF	
JMS	AUXP	/A IMPRIMIRSE UNA ATOMO.
LAC		
PAX		
JHS	PATOM	
LAC	AUXP	/RECUPERO EL VAL. DEL APUNTADOR.
IGUAL	PLISP	
AUXP	0	

RUTINA PUSH2
/PUSH2 HACE UN PUSH DOBLE EMPUJANDO EN EL PDSI
/LOS VALORES COLOCADOS EN LAS DIR. CAR Y CDR.

PUSH2	0	
LAC	CAR	
DAC*	10	
LAC	CDR	
DAC*	10	
JMP*	PUSH2	

CAR	777777	/-1 COMO COND. INICIAL
CDR	777777	/-1 COMO COND. INICIAL

CDR1	0	/LOC. AUXILIAR PARA ALMACENAR EL CDR TEMP.
-------------	----------	--------------------------------------------

RUTINA POP2
/POP2 BAJA LOS VALORES DEL PUSH A LAS LOC. CAR Y CDR.

POP2	0	
LAC	10	/APUNT. DEL PDL APUNTA AL ULTIMO USADO.
TAD	(-2)	
CML		
DAC	10	/NUEVO VALOR DEL APUNTADOR
DAC	11	
LAC*	11	
DAC	CAR	
LAC*	11	
DAC	CDR	
JMP*	POP2	

RUTINA PUSHNO
/PUSHNO ES LA RUTINA QUE HACE UN PUSH CON EL NODO
/APUNTADO POR EL AC Y REG. CON EL VAL. DEL CAR EN EL AC.

PUSHNO	0	
PAX		/AC AL IR
JMS		
JMS	MODON	
LAC	CAR	
JMP*	PUSHNO	

/RUTINA PUSHI
/PUSHI ES UN PUSH CON 23777 DE DIR DE INICIO, ESTE CRECE
/HACIA LOS VALORES MAS PEQUEÑOS SU APUNTADOR ES
/LA LOCALIDAD 12.
/PUSH APUNTA AL PRIMERO LIBRE Y NO AL ULTIMO USADO.

PUSHI	0	
DAC	TEMP	
LAC	12	
TAD	(-1)	
DAC	12	
LAC	TEMP	
DAC*	12	/SE HACE EL PUSH,
LAC	12	
TAD	(-1)	
DAC	12	/HACE QUE APUNTE UN LUGAR ANTES.
JMP*	PUSHI	

/UN POPI SE EJECUTA CON UN LAC* 12
/UN CHECKADOR PARA POPI VACIO MIRARIA LA LOCALIDAD 12
/Y EXIGIRIA QUE ESE VALOR FUERA MENOR IGUAL QUE 23777

/RUTINA POPI

POPI	0	
LAC*	12	
JMP*	POPI	

/RUTINA NODOW
/NODOW SE UTILIZA PARA METER VALORES A LAS LOCALIDADES
/CAR Y CDR UTILIZANDO A IR COMO APUNTADOR DE VA-
/LORES SE UTILIZA EN CONJUNCAON CON PUSH2 Y POP2

NODOW	0	
LAC	0,X	
DAC	CAR	
LAC	1,X	
DAC	CDR	
JMP*	NODOW	

/RUTINA POPU
/POPU ES LA RUTINA QDE PPT ZERO EJECUTA DE POP2
/Y DESPUES HACE UN PUSHNO CO EL CDR CORRECTADO POR EL
/AC, AL REGRESO AC TIENE EL CAR DEL ULTIMO PUSHNO.

POPU @
 DACP AUX2
 JMS POP2
 LAC AUX2
 JMS PUSHNO
 JMP# POPU

AUX2 @

/PUSHNO Y POPO SON UTILIZADAS POR EVAL PARA LLEVAR
/A FIN LA RECURSIVIDAD DE EVAL.

PUSHNO @
 DACP 15
 JMP# PUSHNO

POPO @
 LAC 15
 DACP AUX2
 TAD (-1)
 DACP 15
 CML
 LAC# AUX2
 JMP# POPO

/RUTINA OCTAL6
/IMPRIME POR EL TTY EN OCTAL EL CONTE. DEL AC.

OCTAL6 0
CLLIRAI:
DAC CAR /CAR USADO COMO AUX.
LAW =6
DAC CDR /CDR USADO COMO AUX.
LAC CAR
RTL
RAL /SE ROTAN TRES LUGARES A LA IZQ.
DAC CAR
AND (7)
TAD (260) /SE COMPLETA EL COD. ASCII
JMS PTY
ISZ CDR /INC. EL CONTADOR
JHP .+10
JMP* OCTAL6

/RUTINA ERROR
/MANDA MENSAJE DE ERROR A UN SUPUESTO ERROR LISP

ERROR 0
DAC NODOW /NODOW USADA COMO LOC TEMP.
JMS RTDF /SEPARADOR
LAC ERRO
SNA /VERIFICA QUE ERROR SE TRATA
JMP .+4 /NO ES FUNCION EL ERROR
PAX /AC AL IR
JMS PATOM /SE IMPRIME EL NOMBRE DE LA FUNCION
DZM ERRO
LAC (ERROR=1) /APUNT. DE PAH. *ERROR*
PAX /AC AL IR (APUNTADOR)
JMS PATOM /SE IMPRIME *ERROR*
AXR 3
JMS PATOM
LAC ERROR
AND (77777) /AC=VAL DEL PC
JIS OCTAL6
LAC (24v)
JMS PTY /SE IMPRIME UN ESPACIO
AXR 2 /IR APUNT. A AC
JIS PATOM /SE IMPRIME AC
DAC NODOW /AL AC EL VAL. DEL AC
JIS OCTAL6 /SE IMPRIME EL VAL. DEL AC
JIS COPUSH /CARGA INICIALES DE PUSHES
JHP ERP /MIGRASA EL CONTROL A EVAL.
ERRO 0 /AQUI SE REP. UNA DIR DE FUNC. DE ERROR

100 5640

START	CAF	
	KRA	
JMS	ENLACE	/SE ENLAZA LA MEMORIA (REG. DE ATOMOS)
JMS	VACONS	/COND. INICIALES DE LECTURA
JMS	RTLF	/RETURN-LINE-FEED
JMS	RTLF	/OTRO SEPARADOR
LAC	(LISP=1)	/APUNT. LISP,24K
PAX		/APUNT. AL IR
JMS	PATOM	/SE IMPRIME (LISP,24K)
AXR	3	/IR=R+3 (APUNT DE (I,N,F,N))
JMS	PATOM	/SE IMPRIME (I,N,E,N)
ERR	JMS	RTLF /SEPARADOR

/EVAL EVAL EVAL EVAL EVAL EVAL EVAL

EVAL LAC CAR
SAD (-1) /PRUEBA PARA MIRAR POR PDL VACIO.
JMP EVALR /FDL RESULTO VACIO TRANSFERENCIA A READ
J-S EVALVI /PRUEBA PARA MIRAR POR VALOR VIEJO
JNP VALVIE /FUE VALOR VIEJO

/ESTE PEDAZO MIRA SI EL VAL. EN EL AC (CAR) TIENE ENCENDIDA LA BAND. DE FUNC. PROG.

RTL /ESTO MIRA SI ES FUNCION PROG. YA MARCADA
SPA
JMP EVAL2 /CAR APUNTA A UNA FUNC. PROG. YA MARCADA
RTR /SE SIGUE EXAMINANDO EL VAL. DEL CAR

CERO1 JMS TESTAC /VAL. DEL AC (ATOMO,NODO,CERO)
JMP EVALW /ATOMO
JMP EVALNO /SE TRATA DE EVALUAR UN NODO,
JNP CDR /RECURSIVIDAD PARA EVAL.

EVALW JMS FORMA /PREG. SI ESTE ATOMO ES FORMA LISP.
SOL /LINK# ES FORMA LISP.
JNP EVAL1 /ES FORMA LISP.

/EN ESTE PEDAZO SE MARCA (BIT 3 A 1) EL APUNTADOR
/DE TODO FUNC. PROG. QUE PASE POR AQUI. ESTO ES
/NECESARIO PARA EL FUNCIONAMIENTO DE LAMBDA. A OTROS
/APUNT.(ARG., FUNC. NAQ.)LOS DEJA PASAR A EVAL2

PAX /A IR PARA PHOC. SI ES FUNC. PROG.
J-S PREG /RUTINA QUE PREGUNTA
JNP EVAL2 /RESULTA SER ARG.
JNP EVAL2 /RESULTA SER FUNC. MAQUINA .

/PEDAZO DONDE SE HACE LA DISTINCION
/ENTRE FUNCION LAMBDA Y LAMBDA*.

JFS TENDO /SE EXIGE QUE SEA UN NODO.
JFS CARFO /SE CAR DE ESE NODO AL AC
SAD LADA
JNP EVALPA /FUNCION EN FUNCION DE LA INDIA.
SAD LADA
JNP EVALPAA /FUNCION EN FUNCION DE LA INDIA*.
J-S ERROR /FUNCION QUE NO ESTA BIEN DEF.

LADA RPA /LA INDIA = APUNTADOR
LADAA RPA /LA INDIA* = APUNTADOR

EVALINDA LAC CDR
DAC AUX1 /LOC. DE ALMACENAJE TEMPORAL.
DAC V-X /VALOR DEL ATOMO AL AC
J-S EVALIND /SE EXIGE QUE CAR SEA UN NODO.
J-S PUSH1 /PUSH2, PUSHRO
JNP EVAL+1

/POY ALICEL DSA FUNCION LAMBDA

EVAL0A PAX
 XOR
 DAC
 CLC
 JMS
 JMP
 (1100000)
 CAR
 /ENCIENDE EL BIT DE FUNC. PROG + ATOMOS
 /-1 AL AC
 PUSH1 /MARCA AL PUSH DE ARG. (CAR) M-A LO USA
 EVAL2 /TODO IGUAL

/EVAL1 DA PROCESO A LAS FORMAS LISP CAUSANDO ESTAS SON DETECTADAS EN EL PUSH-DOWN DOBLE.

EVAL1	PAX	CDR	/AC AL IR (ES FORMA LISP)
	LAC	CDR	/ALMACENO EL VALOR DEL CDR
	DAC	CDR1	/NUEVOS VALORES (OTRO NODO)
CEDE	JMS	POP2	/IR AL AC (APUNTADOR DE LA DIR. DE LA FORMA)
	PAX	ERRO	/APUNT. DEL NOMBRE EN CASO DE ERROR
	DAC	(JMP#)	
	TAD	.+2	/DEP. LA INST. DE BRINCO.
	DAC	CDR1	/AL AC EL VAL. ANTERIOR DEL CDR.
	NOP		/EVAL CEDE EL CONTROL A LA FORMA LISP.
EVAL2	LAC	CDR	/A ANALIZAR EL CDR
	JHS	TESTAC	
	JHS	ERROR	/PARA ATOMO PARECE ==ERROR==
	JHP	EVAL3	/FUE NODO.
EVAL5	LAC	CAR	/CDR#0 (ARG. COMPLETOS)
	AND	(77777)	/QUITA LA MARCA SI LA TIENE.
	PAX		/EL CAR A IR PARA SER ANALIZADO POR PPEG.
	JMS	POP2	/NUEVO NODO A DISPOSICION.
	LAC	0,X	/UNA PRUEBA
	SNA		
	JMP	EVALER	/A ERROR FUNCION NO DEFINIDA
	JMS	PPEG	/IR--(ARG,FUNC.,MAQ.,FUNC.,PROG.)
	JMP	EVAL8	/ARGUMENTO.
	JHP	EVAL7	/FUNC. MAQ. (EVAL CEDE EL CONTROL)
EVAL6	JHS	PUSHNO	/METO EL NODO QUE DEF. ESA FUNC.
	JHP	EVAL+1	
EVAL7	DAC	CDR1	
	JMP	CEDE	
EVAL8	DZM	CDR	/CERO AL CDR DEL NODO EN TURNO
	ADD	(377777)	/QUITO LA BANDERA DE ARGUMENTO.
	JMS	PUSH1	/METO EL VAL. AL PUSH DE ARGUMENTOS.
	JHP	EVAL	
EVAL9	JMS	PUSHNO	
	JHS	TESTAC	
	JMP	EVAL9	/ATOMO (PROCESA ARGUMENTOS LIBERALES)
	JHP	EVAL4	/NODO (PROCESA ARG. EN FORMA DE LISTA)
	JHS	ERROR	/PARA CERO AUN NO SE
EVAL4	PAX		/AC AL IR PARA HACER EL PUSH2 DE ABAJO.
	LAC	CDR	
	DAC	CDR1	

	J-S	POP2	
	LAC	CLRL	
	DAC	CDR	
	JMS	PUSH2	
	JMS	NODOW	
	JMP	EVAL	
EVAL9	PAX	/POR AQUI SE PROCESAN ARG. EN FORMA LINEAL.	
	JMS	PREG	
	JMP	.+3	
	JMS	ERROR	
	JMS	ERROR	
	AND	(37/777)	/QUITO LA BANDERA DE ARGUMENTO
	JMS	PUSH1	/EL ARG. AL PUSH DE ARGUMENTOS
	LAC	CDR	
	DAC	AUX	/DEPOSITO TEMP. DE CDR.
	JMS	POP2	/NUEVO NODO A DISPOSICION
	LAC	AUX	
	DAC	CDR	/CAMBIO EL VALOR DEL CDR DE ESTE NODO.
	JMP	EVAL	
EVALER	DZN	ERRO	/NINGUN NOMBRE SE IMPRIMIRA
	JMS	ERROR	/*EVALUACION DE FUNCION NO DEFINIDA».
/EVALNO PROCESA EL CASO EN QUE CAR ES UN NODO.			
EVALNO	PAX	/APUNT. DE CAR A IR.	
	CARNO	/CAR DEL NODO QUE APUNT. EL AC	
	SAD	LADA	/PREGUNTA POR FUNCION LAMBDA.
	JMP	EVLA	
	SAD	LADAA	/PREGUNTA POR FUNCION LAMBDA.
	JMP	EVLA	
	DZN	CAR	/INDICADOR DE RECURSIVIDAD AL PUSH
	LAC	CDR	
	JMS	PUSHO	/ESTO DEBE SER UN PUSH (EN RECURSIVIDAD)
	LAC	(RECUR)	/DIR. DE REGRESO DE RECURSIVIDAD.
	DAC	CDR	
	PXA	/APUNT. AL AC	
	JMS	PUSHNO	/NUEVO NODO.
	JMP	EVAL+1	/EL CONTROL A EVAL.
RECUR	JMS	POPI	/EL VAL. DE LA EVALUACION DE CAR.
	DAC	CAR	/SE FORMARA UN NODO PARA LA EVALUACION.
	JMS	POPO	/CON RECURSIVIDAD DE EVAL).
	DAC	CDR	
	JMP	EVAL	/REGRESA EL CONTROL A EVAL.
/SE PROCESA LA FORMA LAMBDA Y LAMBDA.			
/DE LA FORMA ((LAMBDA (A1 A2) (E1) (B1 B2))			
EVAL	PLX	/IR A IR	
	LAC	CAR	
	DAC	I,X	
	PXA	/IR AL AC	
	I-AC	CAR	
	4XR	4	/INCREMENTA ED 4 IR
	PXL	/IR AL IR	
	JMP	EVAL+1	/REGRESA EL CONTROL A EVAL
EVALR	LAC	12	

SAC (23777)
JNP .+4 /PUSHI ESTA VACIO.
JNS POP1
JNS PUSHSP /SE ESCRIBEN LOS VALORES.
JNP .+5
JNS RTLF
JNS RTGF
JNS RTGF /UNICAMENTE ES UN SEPARADOR.
JMS READ /SE LEE UNA ESTRUCTURA LISP.
JNS PUSHNO /NUEVO NODO A DISPOSICION.
JNP EVAL+1

VALVIE AND (77777)
PAX /AC AL IR
MAC CDR /EL VAL. VIEJO AL AC.
DAC S,X /VAL. VIEJO ES NUEVO AHORA.
JNS POP2 /NUEVO NODO APROCESAR POR EVAL.
JNP EVAL /SE CIERRA EL CONTROL A EVAL.

/RUTINA EVALVI
/EVALVI PRUEBA SI EL AC. TIENE ENCENDIDO EL BIT 1
/SI ES ASI DETECTA VALOR VIEJO
/ABUSADO (NO SE TRATA DE BIT 0
/SALIDA 1 ==> VALOR VIEJO.
/SALIDA 2 ==> SIGUE.

EVALVI 0
RAD
SMA /AC NEGATIVO ES VALOR VIEJO.
ISZ EVALVI /SE PREPARA LA SALIDA 2.
RAR /SE ESTABLECE LEI. VAL. DEL AC
JMP# EVALVI

/DEFINE DEFINE OFFINE DEFINE DEFINE

DEFINE	JPS	TENODO	/PREG. SI EL AC ES NODO
DEFIN0	JPS	PUSHNO	/PUSH Y AC=CAR NUEVO
DEFIN1	JPS	TENODO	/PREG. SI EL AC ES NODO
	JPS	PUSHNO	/PUSH Y AC=CAR NUEVO
	JPS	TEATON	/EXIGE QUE AC APUNT. A UN ATOMO
DEFIN2	DAC	AUX	/ALMACENO EN VALOR DEL CAR.
	LAC	CDR	
	JPS	TENODO	/EXIGE QUE AC APUNT. A UN NODO
DEFIN3	JPS	POP0	/POP2, PUSHNO, AC=CAR DEL ULTIMO PUSHNO,
	DAC	CDR	/SE EXAMINA EL CDR.
	SZA		/SE EXIGE AC#0
	JPS	ERRKOR	
DEFIN4	DAC	CAR	
	JPS	TENODO	/SE EXIGE SEA NODO EL CAR
DEFIN5	DAC*	AUX	/SE HACE EL ENLACE TIPO DEFINE
	JPS	POP2	
	LAC	CDR	
	JPS	TESTAC	
	JPS	ERROR	/ATOMO ---ERROR---
	JNP	DEFIN6	/TODO
DEFIN7	JPS	POP2	
	JNP	EVAL41	/DEFINE HA TERMINADO -CEDE CONTROL A EVAL-
DEFIN8	JPS	POP0	
	JNP	DEFIN1	

/LA BDA LA IRDA LA-BDA LA-IRDA

LAMBDA	JAS	TE-NODO	/SE EXIGE QUE AC TRAIGA UNA DTH DE VALOR.
	JAS	PUSHNO	/UN PUSH CON EL NODO CORRECTAMENTE PUESTO EN AC.
	LAC	CDR	/SE EXAMINA EL VALOR DEL CDR.
	JMS	TENODO	/SE EXIGE QUE CDR=ODO.
	DAC	AUX	/ESTE VALOR SERA USADO AL REGRESO DE LA BDA.
	LAC	CAR	/EL CAR AL AC.
	JMS	TESTAC	/ANALIZO EL VALOR DEL CAR
	JMP	LAMBDA3	/ATOMO -LAMBDA ATOMO-
	JMP	LAMBDA7	/NUDO -LAMBDA LISTA
	JMS	ERROR	/ERRORES DE SINTAXIS

/LAMBDA7 LAMBDA TIENE UNA LISTA DE ARG.

LAMBDA7	JMS	POP0	/POP2,PUSHNO,AC=CAR NUEVO
	JMS	TESTAC	/PRUEBA PARA MIRA LISTA VACIA.
	JMP	.+4	/ATOMO
	JMS	ERROR	/NUDO --ERROR--
	JMS	POP2	/CERO (LISTA VACIA)
	JMP	LAMBDA2	/HACIA LA SALIDA

/LAMBDA TIENE LISTA NO VACIA DE ARG. LAMBDA
 /LOS HACE VIEJOS Y LOS ENLAZA A LOS NUEVOS VALORES.

LAMBDA	JMS	TE-ATOM	/SE EXIGE QUE CAR SEA UN ATOMO.
	PAX		/AC AL IR
	LAC	CDR	/AL AC EL CDR DEL NODO EN TURNO.
	DAC	AUX1	/EL CDR SE ALMACENA TEMPORALMENTE.
	JMS	POP2	
	LAC	,X	/SE OBTIENE EL VAL. QUE SERA VIEJO.
	JMS	VIEJO	/ PUSH PARA QUE EVAL DETECTE ESE VIEJO VAL.
	PXA		/A IR AL AC
	JMS	PUSMA	/ALMACENA EL VAL. DEL AC PARA ENLACE POSTERIOR
	LAC	AUX1	/SE EXAMINARA EL CDR ALMACENADO.
	JMS	TESTAC	/SE ANALIZA EL CDR ALMACENADO TEMPORALMENTE.
	JMS	ERROR	/ATOMO --ERROR--
	JMP	LAMBDA1	/NUDO (SIGUEN VAR. A ENLAZAR.)
	LAC	(3777)	/CERO (PRUEBA SI POPA ESTA VACIO)
	SAD	13	/APUNTADOR DE PUSH
	JMP	LAMBDA2	/POPA (VACIO)
	JMS	POPA	/ATOMO A ENLAZAR. (POPA NO ES VACIO)
	PAX		/AC A IR
	JMS	ENLAZA	/ENLACE TIPO LAMBDA.(PONE N. DE ARG.)
	JMP	.-6	

LAMBDA1	JMS	PUSHNO	
	JMP	LAMBDA	

LAMBDA3	PAX		/AC AL IR
	JMS	POP2	/UN POP PARA QUITAR UN NODO (PROCESADO)
	LAC	,X	/EL VAL. DEL ATOMO AL AC
	JMS	VIEJO	/SE VERIFICA VIEJO ESE VALOR.
	PXA		/EL APUNT. DEL ATOMO AL AC.
	DAC	AUX3	/LOC. TEMP. PARA HACER EL ENLACE DE VALORES
	CLC		/-1 AL AC PARA USARSE COMO MASCARA
	JMS	PUSMA	/SE PONE MARCA AL PUSH AUXILIAR.
	JMS	POPI	/POP AL PUSH DE ARGUMENTOS
	SAD	(-1)	/PREQ. POR MARCA DE PUSH DE ARG.

	JMP	LAMBDA4	/RESULTO LA MASCARA
	JMS	PUSHA	/METO EL ARG. EVALUADO AL PUSH AUXILIAR
	JMP	.=4	/A SACAR OTRO ARG. YA EVALUADO.
LAMBDA4	LAC	PRIVA	/AL AC EL PRIMER NODO LIBRE.
	XOR	(400000)	/PONGO BANDERA DE ARG.
	DAC*	AUX8	/SERIA EL VALOR DE LA VARIABLE LAMBDA
	AND	(77777)	/MASCARA PARA PROCESAR ESTE NODO
	PAX		/AC AL IR (PARA TENERLO DE APUNTADOR)
	AAC	1	/SUMA 1 AL AC
	DAC	AUX8	/AUX8 ES AHORA EL APUNT. DEL CDR
REP	JMS	POPA	/SALE UN ARG. EVALUADO
	SAD	(-1)	/PRUEBA SI ES LA MASCARA DE FIN DE ARG.
	JMP	MARCA	/A CERRAR LA LISTA.
	DAC	0,X	/LO ENLAZA EN LA LISTA DE ARG.
	PXA		/IR AL AC
	AAC	1	
	DAC	AUX8	/AUX8 TIENE EL APUNT. DEL CDR
	LAC	1,X	/AL AC LA DIR DEL PROX. NODO LIBRE.
	PAX		/IR APUNTA AL NUEVO NODO LIBRE
	JMP	REP	/A BUSCAR OTRO ARG. O LA MARCA
MARCA	DZH*	AUX8	/CIERRA LA LISTA
	PXA		/AL AC EL APUNT. AL PRIM. NODO LIBRE.
	AAC	2	
	DAC	PRIVA	/PRIVA DEBE APUNT. A LA PRIM-LOC. LIBRE
	JNP	LAMBDA6	/EL ENLACE ESTA HECHO (FIN LAMBDA)
	/LAMBDA2 PROCESA LA SALIDA DE LAMBDA CUANDO ESTA /TIENE UNA LISTA DE VARIABLES (AUNQUE SEA VACIA)		
	/LAMBDA6 PROCESA LA SALIDA DE LAMBDA CUANDO ESTA /TIENE UNA VARIABLE PARA ENLAZAR.		
LAMBDA2	JMS	POPI	/ESTE VAL. DEBE DE SER LA M-REA.
	SAD	(-1)	/DE LA FUNCION PROGRAMADA.
	SKP		/CORRECTO ES LA MARCA
LAMBDA6	JMS	ERROR	
	LAC	AUX	/PUSHO AL FINAL DE LAMBDA.
	JMS	PUSHO	/SE HACE UN PUSH CON EL NODO CONECTADO POR EL AC.
	JMS	TESTAC	/SE ANALIZA EL VALOR DEL AC =CAR=
	JNP	LAMBO5	/ATOMO =EXPRESION LAMBDA ATOMO.
	JNP	LAMBO8	/NODO =EXPRESION LAMBDA LISTA.
	JMS	ERROR	-----EPOR DE SINTAXIS -----
	/LAMBDA8 DA LA SALIDA A LAMBDA CUANDO SU EXP. /A EVALUAR =S DE LA LISTA.		
LAMBDA8	LAC	CER	
	SZA		/SE EXIGE QUE CDR SEA CERO.
	JMS	SIERRA	/=ERROR DE SINTAXIS DENTRO DE LAMBDA
	LAC	CAR	
	JMS	POP2	/POP2, PUSHO, AC=CAR DEL ULTIMO NODO.
	JNP	EVAL	/LA NODA CDE EL CDR FROZ A EVAL.
	/LAMBDA8 DA LA SALIDA A LAMBDA CUANDO LA EXPRESION /A EVALUAR =S DE LA LISTA.		

LAMBDS	PAX	/AC C APUNTADOR DE /TEN. O AL AC
LAC	PX	/VALOR DEL ATOMO AL AC
SZA		/EXIGE QUE SEA ARG. (INT) O (L)
JPS	ERROR	/*--ERROR-- EL VAL. DFT ARG ID NO ES ARG.
AUD	(77777)	/QUITA LA BAND. DE ARG.
JMS	PUSH1	/UN VAL. AL PUSH DE ARG.
LAC	COR	/CHECA LA SINTAXIS.
SZA	ERROR	/ -----ERRORE DE SINTAXIS-----
JMS	POP2	/SALE UN NODO PROCESADO
JMP	FVAL	
AUX	0	
AUX1	0	
AUX8	0	

/*LAMBDA* LAMBDA* LAMBDA* LAMBDA* LAMBDA* LAMBDA*

LAMBDA	JMS	TE/ODD
	JMS	PUSHNO
	LAC	C/R
	JMS	FE/ODD
	DAC	AUX /PARA LA SALIDA.
	LAC	CAR
	JMS	TESTAC
	JIP	LAMCAA /ATOMO
	JNP	LAMNIN /NODE
	JMS	ERROR /* ERROR **

/*POR AQUI PASA (LAMBDA* D EI)

LAMBDA	PAK	/AC AL IR
	JPS	POP2 /SE QUITA UN NODO YA ANALIZADO.
	LAC	0,X
	JMS	VIEJO
	LAC	AUX1 /VALOR DE CDR ALMACENADO (ARGUMENTO)
	SIA	
	LAC	(24000) /APUNT. DE LISTA VACIA.
	XOR	(40000) /BAND. DE ARGUMENTO.
	DAC	0,X /NUEVO VALOR EN ACCION.
	JIP	SALIDA+1

/*LAMBDA PROCESA (LAMBDA* C ALGO) EI)

LAMBDA	JPS	POPU /QUITA EL NODO YA PROCESADO Y METE EL
	JPS	TESTAC /NODO DEL PRIM. VARIABLE.
	JIP	ATOMO /ATOMO
	JMS	ERRFOR /NODE
	JIP	SALIDA /LISTA VACIA.
	JPS	TEATOM /SE EXIGE CAR APUNTE A UN ATOMO.
	PAK	
	LAC	CDR
	DAC	CDR1 /ALMACENA TEMP. EL CDR
	JPS	POP2 /QUITO UN NODO YA PROCESADO.
	LAC	0,X /EL VAL. DE ESE ATOMO AL AC.
	JPS	VIEJO /SE VALE VIEJO ESE VALOR.
	LAC	AUX1 /APUNT. DE VALORES A ENLAZAR.
	JPS	CAR0 /EL VAL. A SER ENLAZADO
	XOR	(40000) /BAND. DE ARG.
	DAC	0,X /VAL. DE LA VARIABLE.
	LAC	AUX1 /VAL. DE LA VARIABLE A ENLAZAR
	JPS	CDR0 /SE GENERA EL SIG. APUNT. DE VAR.
	DAC	AUX1 /VAL. ENLAZADO.
	LAC	CDR1 /EL CDR ALMACENADO EN AC.
	JIP	SALIDA+1 /SE TERMINAN LAS VARIABLES A ENLAZAR.
	JPS	REGONO /SE EXIGE QUE SEA NODE.
	JPS	PUSHNO /
	JIP	ANEGNO /SE REGRESA ENLAZADO DE VARIABLES.
	JPS	EOF2
	JIP	0/1 16 /SE PONE Y DIBLAR* SABER TIENDA.

ZROTIKA VIEJO
VIEJO SE ENCARGA DE TRASLADAR LOS DATOS DE LOS
ANTERIORES PROCESAMIENTOS DE LA PUA AL C. SE GUARDAN EN
LA BANDERA DE VIEJO.

VIEJO R
JMS PUSH2 /IR AL AC C LA DIR. DEL ATOMO.
PXA /SE PONE LA BANDERA DE JMS. VIEJO.
XOR (ZERODOS)
DAC CAR
LAC D,X /SE OBLIGA EL VAL. DEL ATOMO A SER VIEJO.
DAC COR /ESE VAL SE COLOCA EN EL CTR DEL CODIGO EN PLENO
JMP* VIEJO

ZOLICA ENLAZA
ENLAZA EL PRIMER ARG. DE PUSH CON EL ATOMO APILADO POR
/IR Y SE PONE A ESE VALOR LA BANDERA DE ARGUMENTO.

PUSH R
JMS POP1 /RETENGO EL PRIMER ARG. DEL PUSH DE ARGUMENTOS.
XOR (ZERODOS) /SE LE PONE LA BANDERA DE ARGUMENTO.
DAC D,X /SE ENLAZA EL VALOR ESE CAUCHA DE VAL. AL ATOMO.
JMP* ENLAZA

ZRUTIKA PREG
/PREG ES LA RUTICA QUE PUS EJECUTA SI EL NÚMERO
/COLOCADO EN IR APUNTA
/1.- ARGUMENTO
/2.- FUNCIÓN MÁQUINA
/3.- FUNCIÓN PROGRAMADA
/SAE CON EL VAL. DEL ATOMO EN AC Y EL IR SIN CAMBIO.

PREG	D	
LAC	o,x	/OBTIENGO EL VALOR DEL ATOMO PARA ANALIZARLO.
SPA		/PRUEBO POR LA BANDERA DE ARGUMENTO.
JMP	ARGU	
ADD	(77777)	/EVITA PROC. LA MARCA DE FUNC. PROG.
TAD	(-24000)	/NEG. DE LA DIR. INICIAL DE VACIO.
S/A		/AC>24000 PARA SER FUNCIÓN PROGRAMADA.
PROG	ISZ	PREG
SAU	ISZ	PREG
LAC	o,x	/AL AC EL VALOR DEL ATOMO.
ARGU	JMP*	PREG

ZRUTIKA FORMA
/FORMA ES LA RUTICA QUE DETECTA SI EL CONT. DEL AC QUEDA
/DENTRO EL RANGO [140,200] SE UTILIZA
/ALARMA DETECTAR SI EL AC APUNTA A UNA FORMA
/DEBERÍA ESTAR EN EL RANGO. LIGERAS NO ESTÁ EN EL RANGO.

FORMA	D	
LAC	TE (+)	/ALMACENO EL VAL. DEL AC TEMPORALMENTE.
TAD	(-140)	
SPA		
JMP	FUERA	
TAD	(+64)	
CIA		
S/A		
FUERA	STL	
LAC	TOPP	
JMP	TOP IA	

/QUOTE QUOTE QUOTE QUOTE QUOTE

/QUOTE UNICAMENTE MANEJA UN ARGUMENTO YA SEA
/ ATOMO O LISTA.

QUOTE	JMS	TIENDO	/QUOTE EXIGE AC=ODO
QUOTE	JMS	PUSHNO	/PUSH CON EL NODO APUNT. POR EL AC, AC=CAR
	LAC	CDR	
	SZA		/SE EXIGE QUE CDR SEA CFRC.
	JMS	ERROR	
QUOTE1	LAC	CAR	
	JMS	TESTAC	/SE ANALIZA EL VALOR DEL CAR.
	JMP	.+3	/ATOMO
	JMP	.+2	/NODO
	JMS	ERROR	/CERO --ENFOR--
QUOTE2	JMS	PUSHI	/SE COLOCAS SOBRE UN PUSH DE ARGUMENTOS.
	JMS	POP2	/SE BORRA EL SOBRE EL PUSH.
	JMP	EVAL	/SE TERMINA LA ACTUACION DE QUOTE.

/ENTRADA ENTRADA ENTRADA ENTRADA ENTRADA

/ENTRADA ES LA FORMA QUE DETERMINA EL APARATO
/DE ENTRADA PARA LA RUTINA READ, ADemas CON LA
/OPCION DE EVITAR EL ECO SI SE JUE POR PTR.

ENTR	DAC	AUX2	/APUNT. DE NODO SE ALMACENA
	JMS	CARNO	/EL CAR DEL NODO APUNT.
	JMS	TEATOM	/SE EXIGE CAR SEA APUNT. DE UN ATOMO
	PAX		/AC AL IR
	LAC	.+X	/SE ESPERA UN 59,61,62,63
	SAD	PTRN	/APARATO CERO.
	JMP	PTRLEE	
TTYLEE	LAC	(JMS PTTY)	/CODIGO DE UNA INSTRUCC.
	DAC	INSTRU	
	LAC	(JMS DEETTY)	
DEP	DAC	LEERCHA	/SE DEP. LA RUT. DE LEC. DE CHA
	LAC	INSTRU	/AC=CODIGO DE UNA INSTRUCCION.
	DAC	RETLF	
	DAC	NOP1	
	DAC	NOP2	
	LAC	AUX2	/EL APUNT. DE NODO AL AC.
	JMS	CARNO	
	SZA		/PRUEBA LA SINTAXIS.
	JMS	ERROR	/ERROR DE SINTAXIS
	JMP	EVAL	
PTRLEE	CIA		/OR ENTRE AC Y SWITCH DE CONSOLA
	OAS		/AC DICE SI SE EVITA EL ECO O NO
	STA		
	JMP	.+3	/LO SE EVITA EL ECO
	LAC	(JCP1)	/SF EVITARA EL ECO AL TTY.
	DAC	INSTRU	
	LAC	(JMS DEEPTR)	
	JMP	DEP	
PTR	640000		/APARATO PTR (PARA E TRADA)
INSTRU	J/S	PTTY	/ESTADO INICIAL

/FUTURA TESIAC
/TESTAC ES LA RUTINA QUE PRUEBA EL VALOR DEL AC
/PARA CONTENER (CERO, ATOMO, O UNA DIR. DE VACIO).
/ATOMO --SALIDA1, NODO --SALIDA2, CIFRO --DSALIDA3
/CUALQUIER OTRA POSIBILIDAD HACE HACE QUE PARE EL PRUG.

TESTAC	J
SI A	
JMP	CERO0
TAD	(-100)
SPA	
JMS	ERROR
TAD	(-3700)
SPA	
JMP	ATOMO
TAD	(-20600)
SPA	
JMS	ERROR
TAD	(-37777)
SPA	
JMP	ATOMO
J S	ERROR
ATOMO	TAD
	(4000)
	JMP*
CERO0	TSZ
	TESTAC
	JMP
NODO	NODO+1
	TAD
	163777)
	TSZ
	TESIAC
	JMP*
	TESTAC

/CAR CAR CAR CAR CAR CAR CAR CAR CAR

CARO	JMS	POPI	/DIA POP AL PUSH DE ARGUMENTOS.
	JMS	TENODO	/SE EXIGE QUE APUNTE A UN NODO.
	PAX		/AC AL IR
	LAC	0,X	
	JMS	PUSHI	/DEP. LA EVALUACION DEL CAR AL PUSH DE ARG.
	JMP	EVAL	/CAR TERMINA SU ACTUACION PREGESA A EVAL.

/CDR CDR CDR CDR CDR CUR CUR CUR CDF CDF

CDRO	JMS	POPI	
	JMS	RECODO	
	PAX		
	LAC	0,X	/PRUEBA QUE EL ARG. NO
	SNA		/SEA LA LISTA VACIA.
	JMS	ERROR	/EL ARG. ES LA LISTA VACIA.
	LAC	1,X	
	JMS	TESTAC	/PRUEBA AL VALOR DEL RESULTADO.
	JMS	ERRHOR	/ATOMO -ERROR-
	SKP		/ODIO (UPA LISTA -BIEN-)
	LAC	(24000)	/CERO (ES LA LISTA VACIA).
	JMS	PUSHI	
	JMP	EVAL	/CDR TERMINA SU ACTUACION.

/CONS CONS CONS CONS CONS CONS

CONS	JMS	MEMOR	/AL AC EL PRIMER NODO DISPONIBLE.
	PAX		
	LAC	1,X	/AL AC EL ENLACE DEL NODO
	DAC	PRIVA	/NUEVO NODO LIBRE
	JMS	POPI	/PRIM. ARGUMENTO
	JMS	TENODO	/SE EXIGE SEA NODO (LISTA)
	DAC	1,X	
	JMS	POPI	/SEG. ARGUMENTO.
	DAC	0,X	
	PXA		/IR AL AC
	JMS	PUSHI	/AL PUSH DE ARGUMENTOS EL RESULTADO.
	JMP	EVAL	/CONS TERMINA SU ACTUACION.

/ATOM ATOM ATOM ATOM ATOM ATOM

ATOM	JMS	POPI	
	JMS	TESTAC	
	ATM	*+4	
	NOP	(F)	/NODO
	LAC	(F)	/APUNTADOR DE LA F
	SKP	(V)	
	LAC	(V)	/APUNTADOR DE LA V
	JMS	PUSHI	
	JMP	EVAL	

ZEROPY EQ 17 EG

EV	J-S	POP1
	DAC	AUX5
	J-S	POP1
	SAD	AUX5
	J-P	.+4
	DAC	(F)
	J-S	PUSHI
	J-P	EVAL
	DAC	(V)
	J-P	.-3

/APERTADOR DE F
/APERTADOR DE V

AUX5 0

/PRINT PRINT PRINT PRINT PRINT PRINT PRINT

PRINT	J-S	POP1	/SALE EL ARGUMENTO.
	J-S	PUSHP	/SE IMPRIME EL ARGUMENTO.
	J-S	PUSHI	/SE DEJA TODO IGUAL.
	J-P	EVAL	

/EVAL EVAL EVAL EVAL EVAL EVAL EVAL

EVAL	J-S	POP1	/SALE EL ARGUMENTO
	J-S	IF .EQD	/SILO SEA NODO
	J-S	PUSHI	/SILO EL NODO AL PUSH
	J-P	EVAL	/RETORNO A EVAL

/NULL VUELVE ALLOV SISTEMA DE CONTROL DE CADENA
 /NULL SPA /CHECA PARD.
 JNP null1
 /NULL JMS LENO0 /SE EXIGE CODIGO EN EL AC.
 JMS PUSH0 /UN PUSH CON ESE CODIGO.
 LAC CDR
 SZA
 JMS ERROR /CHECA SINTAXIS.
 LAC CAR
 JMS TESTAC /PRUEBA (EXAMINA EL ARG. DE NULL)
 JMP .+3
 JHP NULL0 /NODO
 JMS ERROR /CERO --ERROR--
 JMS FATOR /EVALUA EL ARG. (FUE ATOMO)
 JMS POP2 /QUITO EL NODO YA PROCESADO
 JMS POP1 /EL ARG. (YA EVALUADO) AL AC
 JMS TESTAC /SE EXAMINA EL ARG. YA EVALUADO
 JMP null2 /ATOMO (NULL ES FALSO)
 SKP /NOTO (PUEDE SER VERDAD NULL)
 JMS ERROR /ERROR (NUNCA SE DEBE EVALUAR ACERO).
 SAD (24*00) /PRUEBA VACIO STANDAR
 JMP null3
 JMS PUSH0 /EXAMINO EL ARG.
 SZA
 JHP null4 /NULL RESULTA FALSO
 LAC CDR
 SZA
 JMP null5
 JMS POP2
 (V) /NULL ES VERDAD
 null5 JMS PUSH1 /T AL PUSH DE ARG.
 JMP EVAL /NULL TERMINA SU ACTUACION.
 null5 JMS POP2 /QUITO UN NODO QUE NO SIRVE.
 LAC (F) /NULL RESULTA SER FALSO.
 JMP null6
 null6 DAC AUX0
 LAC (null) /AL AC EL APUNT. DEL ATOMO NULL
 DAC CDR
 DAC CDR
 CLAICMA /-1 AL AC
 DAC CDR
 LAC AUX0 /TRUMBO A EVALUAR EL ARGUMENTO DE ATOMO
 JMS PUSH0 /QUE SE PRESENTO EN FORMA DE LISTA.
 JMP EVAL

/IF IF
 IF SPA /A GET. (IF0 O IF1)
 JPP IF1
 IF0 JPS PUSHQD
 JPS TESTAC /EXAMINO EL VAL DEL AC (ATOMO NODO CERO)
 JPP IFATOI /ALGO EL CAR.
 JPP IFNODI /CARENODO
 JPS ERROR /CENO ***ERROR**
 IFNODI DAC AUX0 /ALMACENAJE TEMP. DE CAR
 LAC CDR
 JMS TENQDO /CHECA CDR#0 (SINTAXIS)
 XOR (4000000) /SE PONE IF1.
 DAC CDR
 LAC (IFF) /UN IF (APUNT. DEL ATOMO IF)
 DAC CAR
 LAC ALX0
 JPS PUSHQD /MODO DE PREDICADO DE IF
 JPP EVAL /IF0 TERMINA SU ACTUACION.
 IF1 RRD 0377/773 /LE QUITO EL 1 AL IF.
 DAC AUXW
 IF2 JPS POF1 /EXIGE QUE EL ARG. SEA T O F
 SAD (F) /COMP. CON APUNT. DE ATOMO F
 JPP FI /EL ARGUMENTO FUE F
 SAD (V) /COMP. CON APUNT. DE ATOMO V
 SAD JPS ERROR /EL ARGUMENTO NO VALE PARA IF -ERROR-
 T1 LAC AUX0 /EL ARG. ES T
 JMS PUSHQD
 JPS TESTAC /EXAMINA EL VAL. DEL CAR
 JPP IFATOI /ATOMO (RUEBO A EVALUARLO)
 SKP /MODO
 JPS ERROR /ERROR
 LAC CDR /AC=CDR
 JMS ENQDO /CUREJUDO (CHECA SINTAXIS).
 LAC CAR
 PAX /PARA ENCIMAR EL MODO APUNTADO POR IR
 JPS CDR /ENCIMA OTRO NODO A CAR Y CDR.
 JPP EVAL /SALIDA DE IF1 ***T***
 F1 LAC ALX0
 JPS POF1
 LAC CDR
 JPS POF1 /EXIGE CDR#0.
 PAX /PARA EL CDR EL CODIGO APUNTADO POR IR
 JPS POF1 /NO SE MANDA A CAR Y CDR
 SKP /EXIGE CDR#0 (CHECA SINTAXIS).
 JPS POF1
 LAC CDR
 JPS POF1 /EXIGE AL VAL. DEL CAR
 JPP IFATOI /ATOMO (RUEBO A EVALUAR ESTE)
 SKP /MODO
 JPS POF1 /***ERROR***
 PAX /PARA ELICIDAR EL NODO APUNTADO POR IR

	JMS	RETORNAR AL NODO Y CICLAR.
	JIP	ZERCA SUELTA DE IF. -----
IFATOT	JMS	EVALUAR EL ATOMO EN EL AC.
	LAC	ZERCA VARIAS LINEAS PARA COMPARAR.
	DAC	
	AUX	
	JMS	ELIMINAR ESTE NODO YA PROCESADO.
	JIP	ZERCA A SEGUIR EVALUANDO EL IF.
IFATOT	JMS	ZERCAUDAR EL ATOMO COLOCADO EN EL AC.
	JMS	ELIMINAR ESTE NODO YA PROCESADO.
	JIP	EVAL
AUX0	0	ZERO DE ALMACENAJE TEMPORAL

ZROUTINA EATON

ZEATON EVALUA EL ATOMO COLOCADO EN EL AC
ZEXAMINANDO QUE REALMENTE ESTE VALOR RESULTE
ZARGUMENTO Y METIENDOLO EN EL PUSH DE ARGUMENTOS

EATON	0	
	PAX	
	LAC	W,X
	AND	(37777) ZQUITA LA BANDERA DE ARG.
	JMS	PUSHI /METO EL VAL. AL PUSH DE ARG.
	JIP*	EATON

ZONTELA TECNO
ZONA 10 ES LA ROTINA QUE DIBUJA EL VALOR
QUE VA EN EL AC CORRESPONDIENTE AL ATOMO

ESTADO C
TAD (=100)
SPA
JIS ERROR
TAO (=37, e)
STA
JIS ERROR
TAD (4444)
JPS TEATO 1

ZONTELA TECNO
ZONA 11 QUE DIBUJA SI EL VALOR QUE VA EN EL AC ES DIBUJO

ESTADO C
TAO (=2495,)
SPA
JIS ERROR
TAD (=34777)
STA
JIS ERROR
TAD (57777)
JPS TEATO 0

ZONTELA TECNO
ZONA 12 QUE DIBUJA LOS LIBRES DE MEMORIA

SEÑOR C
LAC PINTA VAL AC EL PRI. 5000 LIBRE
TAO LINE 10
STA
JIS +3
LAC PINTA
JPS ERROR
JIS CLEIA ZCOLECTOR DE BASURA
JIS -7

LAC PINTA = 1000 ZONTELA PL. MEMORIA (OPCIO + 0010)

/COLECTOR BASURA COLECTOR BASURA COLECTOR BASURA

COLBA 0

/SE ALMACERAN LOS VALORES DE LOS REGISTROS
/IMPORTANTES (AC , LINK , IR , LR) .

DAC	AUXAC	/ALMACERENO VAL. DEL AC
RAL		
DAC	AUXL	/ALMACERENO VAL. DEL LINK
PXA		/IR AL AC
DAC	AUXIR	/ALMACERENO VAL. DEL IR
PLA		
DAC	AUXLR	/ALMACERENO VAL. DEL LR
JMS	RTLF	/UN SEPARADOR

/SE IMPRIME EL LETRERO 'COLECTOR DE BASURA'
LAC (50) /APUNT. DE LA PAL. 'COLECTOR'.
PAX /AC AL IR
JHS PATOM /SE IMPRIME 'COLECTOR'.
AXR 3 /APUNT. DE LA PAL. 'DE'
JHS PATON /SE IMPRIME 'DE'
AXR 1 /APUNT. DE LA PAL. BASURA
JMS PATOM /SE IMPRIME 'BASURA'.
JHS RTLF /UN SEPARADOR.

/SE MARCA (PARA EL COLECTOR DE BASURA) LAS
/ESTRUCTURAS LIGADAS A LA REGION DE ATOMOS.
/LR=DIR. FINAL DE LOS ATOMOS.

ITERA	DAC	(214)	/DIR. INICIAL PARA ATOMOS.
	PAX		
	AXS	4	/SUMA 4 Y SALTA SI IR>SLR
	SKP		
	JMP	COLBAI	/A OTRO PASO DEL COLECTOR
	LAC	0,X	/SE OBTIENNE EL VAL DEL ATOMO
	AND	(77777)	/SE QUITAN LAS POSIBLES BANDERAS
	JMS	TESTAC	
	JMP	ITERA	/ATOMO
	SKP		/NODO
JMP	ITERA	/CERO	
DAC	S		
JMS	BARCO	/SE MARCA LA ESTRUCT. COLGADA DE UN ATOMO	
JMP	ITERA	/A BUSCAR OTRO ATOMO.	

/SE MARCA (PARA EL COLECTOR DE BASURA)
/LAS ESTRUCTURAS LIGADAS AL PUSH-DOWN DE ARGUMENTOS
/12 ES EL APUNTADOR DEL PUSH APUNTA AL PRIMERO
/LIBRE Y NO AL ULTIMO USADO.

COLBA1	LAC	12	/APUNT. DEL PUSH DE ARG.
	DAC	17	
OTRAA	SAD	(23777)	/FIN DEL PUSH DE ARGUMENTOS
	JMP	COLBA2	/A OTRO PASO DEL COLECTOR
	LAC*	17	
	JHS	TESTAC	/SE EXAMINA EL ARGUMENTO
	JMP	.+5	/ATOMO
	SKP		/NUDO
	JMP	.+3	/CERO
	DAC	S	/SE MARCARA UNA ESTRUCTURA
	JHS	MARCO	
	LAC	17	
	JFP	OTRAA	/RUMBO A ITERAR

/LOCALIDADES DE ALMACENAJE TEMPORAL

AUXAC	0	
AUXL	0	
AUXIR	0	
AUXLN	0	
DIRECT	0	/LOC. DE ENLACE DE NODOS
LSMEMO	40044	/LIMITE SUP. DE MEMORIA (OPCION DDT)

/SE MARCA (PARA EL COLECTOR DE BASURA) LAS
ESTRUCTURAS LIGADAS AL "PROBLEMA-ORIGEN".

COLHA2	JMS	PUSH2	/CAR Y CDR AL PUSH PORTE
	LAC	10	/APUNT. DEL PUSH PORTE
	DAC	LIMITE	/APUNT. AL VAL. CDR DEL PDL DOBBL.
OTR00	LAC	(13777)	/INICIO DE EXAMEN PARA EL PUSH PORTE
	DAC	10	/17 FUNCIONA COMO APUNTADOR
	LAC	17	
	SAD	LIMITE	/PREG. SI ACABA EN PROCESO
	JMP	COLHA3	/A OTRO PASO DEL COLECTOR
	LAC*	17	/PUNTADO EN CAR DEL NODO A EXAM.
	SPA		
	JMP	VAC100	/ES 77777 MARCA DE PUSH VACIO.
	RAL		/VERBENA PARA VAL. VIEJO
	SPA		
	JMP	AT0100	/FUE VAL. VIEJO EL CAR
	RAR		/REGRESAMOS AL VAL. ORG. DEL CAR
	AND	(77777)	/AQUI CAR=ATORO, TODO O CERO
	JHS	LASTAC	
	JAP	ATO 00	/ATORO
	JMP	SOB00	/ATORO
	JMP	CERO0	/CERO
	LIMITE	0	
VAC100	ISZ	17	
	JMP	OTR00+2	
AT0100	LAC*	17	/SALE EL CDR
	SRA		
	JIP	OTR00+2	/ES CERO EL CDR
	AND	(77777)	/QUITA POSIBLE BAND. DE CDR
	DAC	S	/NECESARIO PARA MARCA
	JHS	MARCA	/SE MARCA LA ESTRUCT.
	JIP	OTR00+2	/SE VA SOBRE OTRO NODO
NOD00	DAC	S	
	JHS	"ARCH"	
	JIP	AT0100	/SE VA SOBRE EL CDR.
CERO0	ISZ	17	/APUNT. INCREMENTADO EN 1
	JHS	DUPO	/SALE EL VAL. DE CDR RECURSIVO
	DAC	DIRECT	/LOC. DE ALMACENAJE TEMP.
	JHS	PUSH00	/NO REGRESO
	DAC	DIRECT	/LOC. DE ALMACEN TEMP.
	JIP	ATO100+1	/A MARCAR ESA ESTRUCT.

ZSE MARCAZ LAS LOCALIDADES MARCADAS Y
ZSE DESMARCAN LAS LOCALIDADES DESMARCADAS

COLMAR	J. S.	POP2	ZDEL PUSH SE SACAN CAR Y CDR
	LAC	IS-PEND	ZAL AC UNA DIR. LIMITE DE MEMORIA
	PAL		ZAC AL LR
	LAC	(24004)	ZLOC. DE INICIO DEL CHEQUEO
	PAX		ZAC AL IR
OT GO	LAC	0,X	ZOTORGUE EL VAL. DEL CAR
	SPA		
	JPP	DES-#1	ZEL CAR ESTA MARCADO
	PAX		ZAL AC LA DIR. DE LA PRIM. LOC. LIBRE
	LAC	DIRECT	
	DAC	PRIVA	ZPRIVA TIENE LA PRIM. LOC. LIBRE
REVISA	AXS	+2	ZSUMA 2 Y SALTA SI JRDLR
	SKP		
	JPP	SALIDO	ZSE ACABO DE HACER LA RECOLECCION
	LAC	0,X	ZEXAMINA EL CAR
	SPA		
	JPP	DES-#2	ZEL CAR ESTA DESMARCADO
	DZM*	DIRECT	ZESTAS 5 INSTRUCCIONES DAN
	IS2	DIRECT	ZCERO RESULTADO EL ENLACE
	PAX		ZDE LOS NODOS QUE QUEDARON
	DAC*	DIRECT	ZSE BASURA EN LA RECOLECCION
	DAC	DIRECT	
	JPP	REVISA	ZA REVISAR OTRO NODO.
DESFA1	J. S.	DES-#F	ZSE QUITA LA MARCA
	AXS	2	
	J. P.	0.1.0	
	HIT		ZNO HAY NODO LIBRE
DESFA2	JPS	DESMAR	
	JPP	REVISA	
			Z(COLECTOR DE BASURA)SE RECUPERAN LOS VALORES
			ZDE LOS REGISTROS ESPECIALES.
SALIDA	JPS	COL-#P	ZESTABLECE COND. TEC. ALTERADAS
	LAC	ACTIR	ZRECUPERA VAL. DE IR
	PAX		
	LAC	0,X#P	ZRECUPERA VAL. DE LR
	PAL		
	LAC	0,X#L	
	RAL		
	LAC	AL-#C	ZRECUPERA VALOR DE TELK
	J. P.	COL-#A	ZRECUPERA VAL. DEB AC.

/RUTINA QUE MARCA o A ESTRUCTURA CERO
/DIRECCION DE LA CICLO LINEA MARCAR ESTRUCTURA
/EN LA LOCALIDAD S

MARCO	O	COMENTARIO
	LAC	10 /APUNTO DE PUSH
	DAC	P /LOC. ESPECIAL USADA POR SIG
	DAC	S /LA DIR. SIG. A MARCARSE
	SNA	/PRUEBA SI S ES CERO
	JMP	BFI / /PRUEBA BAND. DE FIN DE ESTRUCTURA
	JHS	MARCAA /RUTINA QUE MARCA EL EXPEDIO S
	JHS	SIG /SE GENERA EL SIG. ELKA. DE LA ESTRUCTURA
	JMP	,=S /SE REPITE EL PROCESO
BFIN	LAC	BTFPRI
	SAD	(3) /PRUEBA BAJERA DE FIN DE ESTRUCTURA
	JMP*	MARCA
	JMP	,=S /AQUI SE TERMINA LA ESTRUCTURA.

/RUTINA PARA APAGA UN RODO

MARCO	O	COMENTARIO
	LAC*	S
	SIA	/PRUEBA LA MARCA
	XOR	(Avs W/D) /VER BIT MARCA ES EL BIT CERO
	DAC*	S /RODO YA MARCADO
	JMP*	MARCAA

/RUTINA DESATAR QUITE LAS MARCAS A LOS RODOS MARCADOS
/EL RODO ES APUNTADO POR IN
/Y LA MARCA UNICAMENTE SE REFLEJA SOBRE EL CAR

DESAR	O	COMENTARIO
	LAC	0,X
	AND	(777777) /SE QUITA LA MARCA
	DAC	1,X
	JMP*	DESATAR

/NOTA: SON DIFERENTES LAS MARCAS EN EL PUSH-DORDE
/Y LAS MARCAS EN LAS ESTRUCTURAS

/BIT 0 -- ESTA LIBRE
/BIT 2 -- FUNCION PROGRAMADA YA MARCADA
/BIT 1 -- TADOR VIF00

BIBLIOGRAFIA

- 1) T. A. BRODY; Symbol-Manipulation Techniques.
Gordon and Breach, Science Publishers (1968).
(este libro contiene una bibliografia bastante extensa sobre el lenguaje LISP).
- 2) CLARK WLISSMAN; Lisp 1.5 Primer. Dickenson Publishing Company, Inc., Belmont, California (1968)
- 3) JOHN McCARTHY; Lisp 1.5 Programmer's Manual, The MIT Press Cambridge, Mass. (1962)
- 4) DANIEL G. BOBROW y otros; The Programming Language Lisp:
Its Operation and Applications.
The MIT Press. (1967)
- 4) HAROLD V. McINTOSH; Operators for MBLISP. Program Note #9.
ALBERTO VERJOVSKY; Two Lisp Pattern Recognition Functions:
Similar and Similar*. Program Note #2.
JOSE BARBERAN; Lisp Function to Calculate Derivatives and
Poisson brackets. Program Note #3.
ROBERT YATES; Group Analysis Programs. Program Note #5.

Quantum Theory Project, University of Florida.
Gainesville, Florida. (1963)
- 5) J. M. FOSTER; List Processing. Macdonald & Co. Ltd. (1967)
- 6) Reference Manual For PDP-15 Systems.
- 7) S. C. KLEENE; Introduction to Metamathematics, Van Nostrand, Princeton, N. Y. (1952)