

Ausiello

Libreria
-15p
★

I A C

ISTITUTO PER LE APPLICAZIONI DEL CALCOLO

DEL

CONSIGLIO NAZIONALE DELLE RICERCHE

PUBBLICAZIONI

Serie III - N. 19

GIORGIO AUSIELLO - CORRADO BÖHM

PETER DEUTSCH - ALFONSO MIOLA - HUGO MONCAYO LOPEZ

Manuale per il sistema ROMALISP

PD 11219

ROMA
1969

DDC
RECEIVED
JAN 12 1971
RECEIVED
B

ACCESSION FOR		
CFSTI	WHITE SECTION	☒
DOC	DDF SECTION	☐
UNANNOUNCED		☐
JUSTIFICATION.....		
.....		
BY		
DISTRIBUTION/AVAILABILITY CODES		
DIST.	AVAIL. and/or	SPECIAL
1	21	

Prof. Dr. H. Stoyan
Universität Erlangen-Nürnberg
Institut für Mathematische Maschinen
und Datenverarbeitung (Informatik VIII)
Am Weichselgarten 9
91058 Erlangen

I A C

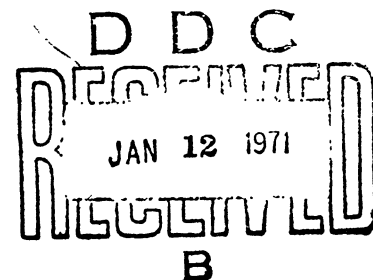
ISTITUTO PER LE APPLICAZIONI DEL CALCOLO
DEL
CONSIGLIO NAZIONALE DELLE RICERCHE

PUBBLICAZIONI

Serie III - N. 19

GIORGIO AUSIELLO - CORRADO BÖHM
PETER DEUTSCH - ALFONSO MIOLA - HUGO MONCAYO LOPEZ

Manuale per il sistema ROMALISP



ROMA
1969

Tipo-litografia MARVES
Roma - Via Mecenate, 35 - Tel. 730.061

NOTA

*Gli Autori ringraziano E. Piccirilli e C. Cecchetti
per l'oneroso lavoro di perforazione dei programmi..*

I N D I C E

INTRODUZIONE	Pag. 7
Cap. 1.- DATI E OGGETTI	" 9
1.1 Premessa	" 9
1.2 Dati	" 9
1.3 Oggetti e strutture	" 10
Cap. 2.- L'INTERPRETE	" 12
2.1 La funzione PROCESS	" 12
2.2 L'entrata dei dati	" 13
2.3 L'uscita dei dati	" 14
2.4 La valutazione delle forme	" 14
Cap. 3.- FUNZIONI	" 16
3.1 Funzioni primitive e compilate	" 16
3.2 Funzioni speciali	" 28
3.3 Funzioni lambda-definite	" 30
3.4 Funzioni ricorsive e funzioni iterative	" 32
- APPENDICI	" 35
A - Definizione LISP dell'interprete	" 35
B - BASIC SYSTEM	" 43
C - Segnalazioni di errore	" 58
D - OPERATE	" 59
- INDICE ANALITICO	" 65

INTRODUZIONE

Il LISP è un linguaggio per l'elaborazione di liste che ha importantissime applicazioni in problemi di manipolazione di simboli (come il trattamento di espressioni algebriche, il calcolo simbolico, la rappresentazione di sistemi formali) e in tutto l'insieme di tecniche attinenti la cosiddetta "Intelligenza artificiale".

La sua prima realizzazione per la IBM 7090 è opera di un gruppo costituito da John Mc Carthy e da altri ricercatori del MIT (1962) ma negli ultimi sei anni la famiglia di interpreti e compilatori LISP si è sviluppata e ramificata enormemente.

Il ROMALISP è un sistema realizzato presso l'I.A.C. del C.N.R. per il calcolatore O.G.E. 9104 (CINAC); esso consiste di un interprete LISP e di un sistema operativo speciale (OPERATE) entrambi codificati in linguaggio ottale non spiazzabile.

La famiglia alla quale il ROMALISP appartiene è quella che ha generato i compilatori MBLISP (IBM 709) e LISP 2(Q-32).

Sotto la supervisione di Corrado Böhm, Hugo Moncayo Lopez del Centro Nazionale di Calcolo di Città del Messico ha disegnato il progetto originario (ROMALISP 1) e ne ha realizzato la parziale codifica durante il 1966. Il ROMALISP 1, ultimato, messo a punto e corredato di un sistema operativo da Giorgio Ausiello, è divenuto operante dal Febbraio 1967. Nel Settembre dello stesso anno, raccogliendo alcuni suggerimenti di Peter Deutsch dell'Università di California Giorgio Ausiello e Alfonso Miola realizzavano una seconda versione dell'interprete (ROMALISP 2) e del sistema operativo, impostati su un meccanismo iterativo di valutazione delle espressioni, la quale diveniva operante dal Gennaio 1968.

Il presente manuale è stato redatto da Giorgio Ausiello e Alfonso Miola.

Sebbene si sia cercato di rendere esauriente questo testo è opportuno precisare che esso non è altro che un manuale per l'utente del sistema ROMALISP e perciò risulterà del tutto chiaro solo a chi sarà già introdotto alla programmazione LISP.

Una buona introduzione al linguaggio LISP è fornita in:

- Clark Weisman, LISP 1.5 Primer, Dickenson Press 1967.

Altri testi fondamentali per la conoscenza del LISP e delle sue applicazioni sono:

- John Mc Carthy: LISP 1.5 Programmer's Manual, M.I.T., Cambridge 1962.

- Berkeley and Bobrow (editors) The Programming Language LISP,

I.I.I., Cambridge 1964.

Per poter conoscere l'organizzazione dei dati e i criteri con i quali l'interprete è stato realizzato sulla CINAC è necessario prendere visione di:

- C. Böhm, H. Moncayo Lopez, G. Ausiello: Costruzione di un interprete CINAC per il linguaggio LISP (ROMALISP 1), Quad. n.10 I.A.C. 1967.
- G. Ausiello, A. Miola: Costruzione di un "garbage collector" per l'interprete ROMALISP, Nota interna I.A.C. 1968.

A tali articoli è opportuno riferirsi per ciò che concerne la gestione della memoria, la nozione di dato, di oggetto e di struttura, e la nozione di funzione primitiva e compilata.

Alcune note rivolte a chi non sia completamente introdotto al LISP sono comunque inserite nel testo (racchiuse tra due coppie di asterischi) e possono essere omesse da chi abbia familiarità con i concetti di programma LISP e di funzione LISP.

Cap. 1 - DATI E OGGETTI

1.1. PREMESSA

L'alfabeto ROMALISP è costituito dai seguenti caratteri CINAC:

- alfabetici ABCDEFGHIJKLMNOPQRSTUVWXYZ
- numerici: 0 1 2 3 4 5 6 7 8 9
- speciali: [[« » , : ; > < ≥ ≤ ≠ + - * / ↑ ∪ ∧ ≡] ∘.
- separatori: () b⁽¹⁾
- neutro: →⁽²⁾

Si chiama stringa una sequenza linearmente ordinata di caratteri.

1.2. DATI

Gli enti che sono a disposizione di un programmatore per redigere un programma ROMALISP sono chiamati "dati".

- *Numero*: una stringa costituita da caratteri numerici o dal carattere - seguito da caratteri numerici e rappresentante un numero intero compreso fra -2.097.152 e 2.097.151. Lo zero è rappresentato da un qualunque numero le cui cifre numeriche siano tutte costituite dal carattere 0.
- *Atomo*: una stringa di non più di 20 caratteri, non contenente separatori e non costituente un numero.
- *Lista*: una stringa iniziante con (e terminante con), contenente un numero maggiore o uguale a 0 di stringhe che sono numeri, atomi o liste, separati da 0, 1 o più spazi, con l'unica condizione che atomi o numeri consecutivi devono essere separati da almeno uno spazio.

Es.: QUEST'E'UNATOMO è un atomo
 (QUESTA b E' (UNA b LISTA)) è una lista
 1-1-1969 è un atomo
 -111969 è un numero
 NEW b YORK non è un dato.

(1) Con il carattere b si indicherà, ove occorra, una occorrenza del carattere spazio.

(2) Il carattere neutro non è rilevato dalle unità di entrata CINAC.

1.3. OGGETTI E STRUTTURE

Con il nome di oggetti si designano nel ROMALISP gli enti sui quali opera il sistema ROMALISP. essi vengono generati all'interno o come codificazione ottale dei dati alfanumerici forniti in entrata o come risultato della elaborazione di oggetti preesistenti forniti come argomenti.

Alla 1^a classe appartengono gli oggetti lista, atomo, numero, alla 2^a, oltre agli oggetti della 1^a classe, anche tavole e valori di verità. Questi ultimi due tipi di oggetti non utilizzabili in entrata come dati sono indispensabili in fase di elaborazione e perciò l'interprete è provvisto di funzioni capaci di generarli e di manipolarli. Inoltre, mentre delle tavole non è consentita neanche l'uscita, i valori di verità sono forniti in uscita con i nomi TRUE (vero) e FALSE (falso).

La rappresentazione di un oggetto è una parola CINAC (otto caratteri ottali) la quale può contenere l'oggetto stesso o rimandare alle strutture dell'oggetto.

- Oggetti che rappresentano se stessi:

- . numero: una parola il cui primo carattere è 0 se il numero è positivo, 7 se il numero è negativo e gli altri sono la codificazione ottale del numero.

- . valore di verità: una parola convenzionale (37000000) per il valore TRUE o una (40000000) per il valore FALSE.

- Oggetti dotati di struttura: una parola i cui primi due caratteri sono una etichetta che caratterizza l'oggetto e i cui ultimi sei caratteri sono l'indirizzo della struttura;

- . struttura di lista: una parola doppia (nodo di lista); la prima parola è l'oggetto che è il primo elemento della lista, la seconda è la lista costituita dai restanti elementi, oppure lista vuota;

- . struttura di tavola (ad n righe): $n + 2$ parole consecutive; la prima fornisce l'indirizzo della tavola preceduto da una etichetta, la seconda contiene la lunghezza (il numero n), le altre n parole contengono oggetti o stringhe di quattro caratteri;

- . struttura di atomo: una tavola a 2 righe; la prima contiene la tavola del nome dell'atomo, la seconda l'oggetto che è il valore dell'atomo; la tavola del nome ("printname") dell'atomo è una tavola le cui righe contengono la stringa di caratteri (quattro per riga, fino a un massimo di cinque righe) che costituisce l'atomo, eventualmente seguita da un numero di caratteri neutri necessari a completare la ultima riga. Per convenzione il valore di un atomo è l'atomo stesso se non è stato esplicitamente definito diversamente.

Esempio: (I.A.C. (DICEMBRE 1968))

e sia 12 il valore dell'atomo DICEMBRE (cioe' 14 ottale)

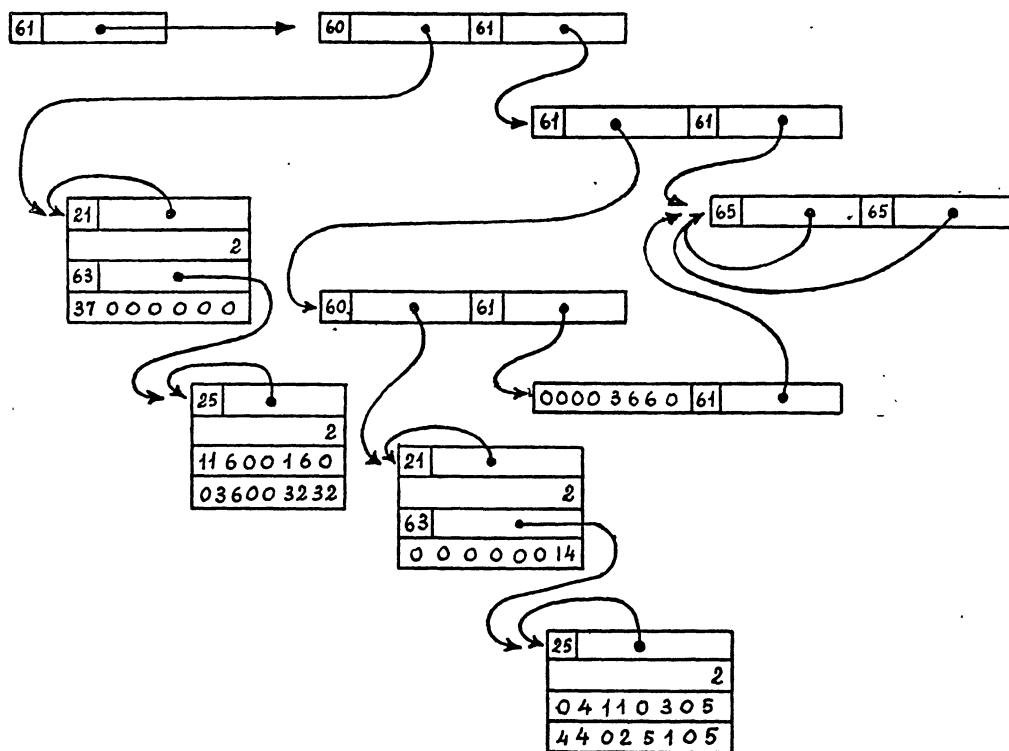
TRUE il valore dell'atomo I.A.C.

116001600360 la codificazione CINAC del nome I.A.C

0411030544025105 la codificazione CINAC del nome
DICEMBRE

3660 la rappresentazione ottale del numero 1968 e

32 la codif. CINAC del carattere neutro



Cap. 2 - L' INTERPRETE

La definizione formale del comportamento dell'interprete ROMALISP 2 è fornita in appendice ed è essa stessa espressa mediante il linguaggio LISP. In questo paragrafo viene invece fornita una descrizione rigorosa ma informale dell'interprete al fine di facilitare la comprensione del funzionamento. Per comodità di espressione le varie funzioni dell'interprete sono contrassegnate con nomi (gli stessi usati per i programmi che costituiscono la descrizione formale).

2.1. LA FUNZIONE PROCESS

Il ciclo fondamentale dell'interprete ROMALISP (funzione PROCESS) è così costituito:

1. entrata di un dato (lista, atomo o numero) (funzione READIN),
2. stampa del dato stesso (funzione PRINT),
3. se il dato è una lista non vuota e non inizia con la parola COMMENT viene considerato come una forma funzionale da valutare (funzione APPLY) e il risultato della valutazione viene stampato (funzione PRINT), altrimenti non viene effettuata alcuna valutazione,
4. reinizializzazione (ritorno al punto 1).

**In virtù di questo procedimento ciclico e della possibilità di modificare lo stato interno I_i dell'interprete (cambiando il valore degli atomi) si può dire che la relazione funzionale fra i dati in entrata E_i e dati in uscita U_i è la seguente:

$$\begin{array}{ll} I_0 & = \text{ sistema LISP vergine} & U_1 & = \text{ val } (E_1, I_0) \\ I_1 & = \text{ eff } (E_1, I_0) & U_2 & = \text{ val } (E_2, I_1) \\ \dots\dots\dots & & \dots\dots\dots & \\ I_{n-1} & = \text{ eff } (E_{n-1}, I_{n-2}) & U_n & = \text{ val } (E_n, I_{n-1}) \end{array}$$

L'uscita dell'interprete in corrispondenza dell'entrata n -sima è il risultato della valutazione dell'entrata stessa ma può essere funzione anche delle $n-1$ entrate precedenti perché esse, modificando il valore di alcuni atomi modificano il si-

stema LISP e influenzano il risultato della valutazione. Ciò è particolarmente interessante se gli atomi modificati sono nomi di funzioni e ciò che si modifica è la loro lambda-definizione e, in effetti, una delle tipiche strutture di un programma LISP è costituita da un alternarsi di definizioni di funzioni e di applicazioni delle funzioni definite a dati.

In generale, quindi, la valutazione di un dato in entrata presenta due risultati: il valore, che, a sua volta, è un dato e viene fornito in uscita e l'effetto, che consiste nel modificare il sistema modificando il valore di alcuni atomi.**

2.2. L'ENTRATA DEI DATI

L'entrata dei dati è effettuata dalla funzione READIN che ha il compito di costruirne in memoria la rappresentazione ed eventualmente la struttura.

La prima operazione che viene eseguita è la lettura di un dato cui segue la scansione dei caratteri che lo costituiscono:

- se si incontra una parentesi aperta si avvia il procedimento di costruzione della struttura di una lista (funzione READLIST);
- se si incontra parentesi chiusa si dichiara che la lista che si sta costruendo è terminata e quindi che la funzione READLIST (l'ultima funzione READLIST, nel caso che la lettura di un nido di liste abbia determinato una chiamata ricorsiva) è conclusa;
- se si incontra uno spazio lo si ignora e si prosegue la scansione;
- se si incontra un carattere alfabetico, numerico o speciale si avvia il procedimento di costruzione di una stringa atomica (funzione READATOM).

Costruzione di una stringa atomica: il carattere che ha innescato la funzione READATOM viene memorizzato e, appresso a lui, tutti i caratteri che si incontrano, continuando la scansione dell'entrata, fino al primo spazio o alla prima parentesi. A questo punto:

- tutti i caratteri che costituiscono la stringa atomica vengono ordinati nelle righe di una apposita tavola,
- se la stringa atomica ha le caratteristiche di un numero, ne viene costruita la rappresentazione ottale,
- altrimenti viene esaminato il registro degli atomi (BUCKET) per vedere se un atomo dello stesso nome è già stato acqui-

sito e in caso negativo ne viene costruita la struttura (valore e printname).

Costruzione di una lista: la scansione dell'entrata viene proseguita, costruendo tutti gli oggetti che vengono incontrati come elementi della lista (atomi, numeri o ancora liste) e costruendo i rispettivi nodi di lista fino a che l'incontro della parentesi chiusa corrispondente a quella aperta che aveva avviato il procedimento non dichiara conclusa l'operazione di costruzione della lista.

N.B. Risultato delle operazioni di costruzione è l'oggetto costruito.

2.3. L' USCITA DEI DATI

L'oggetto di cui si richiede la stampa viene esaminato e

- se è un atomo, la stringa di caratteri che ne costituisce il nome (e che si trova nella tavola del printname dell'atomo) viene trascritta in una apposita zona di memoria (buffer di uscita),
- se è un numero viene convertito in decimale e i caratteri numerici vengono trascritti nel buffer di uscita,
- se è un valore di verità il printname dell'atomo TRUE o dell'atomo FALSE viene trascritto nel buffer di uscita,
- se è una lista vengono trascritti nel buffer, il carattere (tutti gli oggetti che costituiscono gli elementi della lista, con l'inserimento di spazi ovunque necessario, e il carattere),
- in qualunque altro caso viene trascritto il printname dell'atomo **** (vedi segnalazioni di errore).

2.4. LA VALUTAZIONE DELLE FORME

Ogni lista che viene fornita all'interprete come dato e che non inizia con COMMENT, viene considerata una forma da valutare, cioè il primo elemento viene considerato una funzione, i restanti elementi gli argomenti. La funzione APPLY ha il compito di dichiarare che gli argomenti della forma entrata non devono essere valutati (anteponendo ad ognuno di essi la funzione QUOTE) e di consegnare la forma così modificata alla funzione EVAL che ne effettua la valutazione.

La funzione EVAL effettua il passo fondamentale (ricorsivo) di valutazione e ovunque, nel seguito, useremo la parola

valutazione di un oggetto intenderemo che a tale oggetto si deve applicare la funzione EVAL. L'argomento di essa può essere un atomo, un numero, un valore di verità o una lista: la funzione EVAL ne fornisce il valore. Tenuto presente che il valore di un atomo è l'oggetto ad esso associato come valore per definizione (vedi struttura di un atomo) e che il valore di un numero, di un valore di verità e della lista vuota è l'oggetto stesso, è interessante definire il valore di una lista non vuota.

- Se il primo elemento della lista è un atomo si presentano le seguenti possibilità:

1. è il nome di una funzione primitiva o compilata,
2. è uno dei nomi chiave AND, OR, IF, COND, LIST, QUOTE,
3. è uno dei nomi chiave PROG, GO, RETURN,
4. ha se stesso come valore,
5. non rientra in nessuna delle categorie precedenti.

Nel caso 1 viene effettuata la valutazione di tutti gli argomenti (secondo, terzo, ..., ultimo elemento della forma funzionale) e ai loro valori viene applicata la funzione (primitiva o compilata) che è stata chiamata; in tal caso l'effetto e il valore che si ottengono sono quelli descritti in 3.1.

Nel caso 2 la valutazione degli argomenti e dell'intera forma si svolge come descritto in 3.2.

Il caso 3 indica che stiamo valutando una forma dotata di "Program Feature" cioè una forma con caratteristiche di programma per la quale si veda il 3.4.

Il caso 4 indica che l'atomo non è stato definito come funzione (vedi segn. err.).

Nel caso 5 viene valutata la forma funzionale ottenuta da quella data sostituendo all'atomo che ne è primo elemento il suo valore.

- Se il primo elemento della lista è ancora una lista si presentano le seguenti possibilità:

1. essa inizia con LAMBDA: è, cioè, una funzione lambda-definita,
2. essa non inizia con LAMBDA.

Nel caso 2 tale lista viene a sua volta considerata una forma funzionale e quindi valutata e, successivamente, il suo valore viene sostituito ad essa come primo elemento della forma data per poi riprendere dall'inizio l'operazione di valutazione della forma data.

Nel caso 1 la valutazione della forma data viene effettuata come descritto in 3.3.

Cap. 3 - FUNZIONI

Le funzioni disponibili per la redazione di un programma ROMALISP con le quali, cioè, si può realizzare la manipolazione di simboli si possono classificare in vari modi: a seconda degli oggetti o delle strutture su cui operano, a seconda del metodo con cui sono state definite, a seconda del meccanismo che l'interprete usa per valutare le rispettive forme funzionali.

La classificazione che noi useremo utilizza questi tre criteri a diversi livelli: al primo livello distingueremo tra funzioni definite all'interno dell'interprete e funzioni definite dall'esterno mediante altri programmi LISP; in secondo luogo distingueremo ancora le prime in primitive (definite con un programma in linguaggio di macchina), compilate (definite in LISP e compilate a mano⁽¹⁾), speciali (caratterizzate da un nome chiave in base al quale l'interprete esegue una valutazione di tipo speciale) e le seconde in ricorsive e iterative; infine avremo la classificazione per tipi di oggetti manipolati.

3.1. FUNZIONI PRIMITIVE E COMPILATE

L'elenco che segue è stato suddiviso per tipi di oggetti manipolati dalle funzioni. Per ogni funzione, accanto al nome, si trova il modello della forma funzionale nella quale la funzione deve essere collocata; in tale modello i nomi simbolici usati al posto degli argomenti indicano a che tipi di oggetti la funzione deve essere applicata, cioè a che classe di oggetti devono appartenere i valori degli argomenti (se la forma deve essere valutata dalla funzione EVAL) o gli argomenti stessi (se la forma deve essere valutata attraverso APPLY la quale inibisce la valutazione degli argomenti).

Alcuni dei nomi simbolici hanno un significato limitato alla funzione descritta e pertanto saranno definiti dove necessario; altri sono usati in generale e sono definiti all'inizio una volta per tutte:

(1) Tale procedimento di definizione e il rispettivo metodo di valutazione si rendono indispensabili per definire in linguaggio di macchina una funzione ricorsiva.

obj : un oggetto (atomo, numero, lista, tavola, valore di verità)
dato : un dato (atomo, numero, lista)
lista, *atomo*, *num*, *logic*, *tab* : rispettivamente una lista, un atomo, un numero, un valore di verità, una tavola.

Gli esempi indicati corrispondono a programmi ROMALISP effettivamente funzionanti: poiché la loro valutazione avviene attraverso APPLY le funzioni vengono applicate direttamente agli argomenti non valutati.

3.1.1. Predicati

TRUE : (TRUE) ha come valore il valore vero;
FALSE : (FALSE) ha come valore il valore falso;
NOT : (NOT *logic*) ha come valore il valore vero se *logic* è il valore falso, il valore falso se *logic* è il valore vero;
ATOM : (ATOM *obj*) ha come valore il valore vero se e solo se *obj* è un atomo;
LIS : (LIS *obj*) ha come valore il valore vero se e solo se *obj* è una lista;
NULL : (NULL *obj*) ha come valore il valore vero se e solo se *obj* è la lista vuota;
NUM : (NUM *obj*) ha come valore il valore vero se e solo se *obj* è un numero;
ARR : (ARR *obj*) ha come valore il valore vero se e solo se *obj* è una tavola;
TRUVAL : (TRUVAL *obj*) ha come valore il valore vero se e solo se *obj* è un valore di verità;
PRIM : (PRIM *obj*) ha come valore il valore vero se e solo se *obj* è il nome di una funzione primitiva;
CSUBR : (CSUBR *obj*) ha come valore il valore vero se e solo se *obj* è il nome di una funzione compilata;
EQ : (EQ *obj*₁ *obj*₂) ha come valore il valore vero se e solo se *obj*₁ e *obj*₂ sono lo stesso oggetto (N.B. *obj*₁ e *obj*₂ non possono essere semplicemente copie di uno stesso dato, ma devono avere la stessa rappresentazione ottale);
ARREQ : (ARREQ *tab*₁ *tab*₂) ha come valore il valore vero se e solo se *tab*₁ e *tab*₂ hanno la stessa lunghezza e contengono gli stessi oggetti (vedi N.B. precedente);
SL : (SL *num*₁ *num*₂) ha come valore il valore vero se e solo se *num*₁ < *num*₂;

LE : (LE num_1 num_2) ha come valore il valore vero se e solo se $num_1 \leq num_2$;
ZERO : (ZERO *obj*) ha come valore il valore vero se e solo se *obj* è il numero zero;
POSV : (POSV *obj*) ha come valore il valore vero se e solo se *obj* è un numero positivo o nullo, oppure è il valore di verità vero;
NEGV : (NEGV *obj*) ha come valore il valore vero se e solo se *obj* è un numero negativo, oppure un atomo, una lista, una tavola, il valore di verità falso.

3.1.2. Manipolazione di liste

CAR : (CAR *lista*) ha come valore il primo elemento della lista argomento;
es.
(CAR (A B C))
A
(CAR ((A B)(C)))
(A B)
N.B. (CAR ())
()

CDR : (CDR *lista*) ha come valore la lista costituita da tutti gli elementi della lista argomento, tranne il primo;
es.
(CDR (A B C))
(B C)
(CDR ((A B)(C)))
((C))
N.B. (CDR ())
()

CxxR CxxxR: dove x sta per una A o per una D; il valore è quello ottenuto componendo, in tutti i modi possibili le funzioni CAR e CDR.

Es.
(CAAR ((A B) C))
A
(CDAR ((A B)C))
(B)
(CADDR (A B C))
C
(CDAAR (((A) B) C))
()

- CONS : (CONS *obj lista*) ha come effetto la costruzione di un nuovo nodo di lista; ha come valore la lista ottenuta aggiungendo *obj* come primo elemento della lista argomento;
- es.
(CONS A (B C))
(A B C)
(CONS (A B) (C))
((A B) C)
L abbia come valore (A B C)
(EVAL (CONS (CAR L)(CDR L)))
(A B C)
- QAR : (QAR *obj lista*) ha come effetto la modifica fisica della lista argomento in quanto sostituisce il CAR del suo primo nodo con *obj*; ha come valore la lista ottenuta sostituendo *obj* al primo elemento della lista argomento;
- es.
(QAR A (B C))
(A C)
(QAR (A) (B (C)))
((A)(C))
- RAR : (RAR *obj lista*) ha lo stesso effetto di QAR e come valore *obj*;
- es.
(RAR A (B C))
A
- SAR : (SAR *obj lista*) ha lo stesso effetto di QAR e come valore il valore vero;
- es.
(SAR A (B C))
TRUE
- XAR : (XAR *obj lista*) ha lo stesso effetto di QAR e come valore il primo elemento della lista argomento;
- es.
(XAR A (B C))
B
- QDR : (QDR *lista₁ lista₂*) ha come effetto la modifica fisica di *lista₂* di cui sostituisce il CDR del primo nodo con *lista₁* e come valore la lista ottenuta sostituendo il CDR di *lista₂* con *lista₁*;
- es.
(QDR (A B) (C D))
(C A B)

RDR : (RDR $lista_1$ $lista_2$) ha lo stesso effetto di QDR, e come valore $lista_1$;
 es.
 (QDR (A B) (C D))
 (A B)

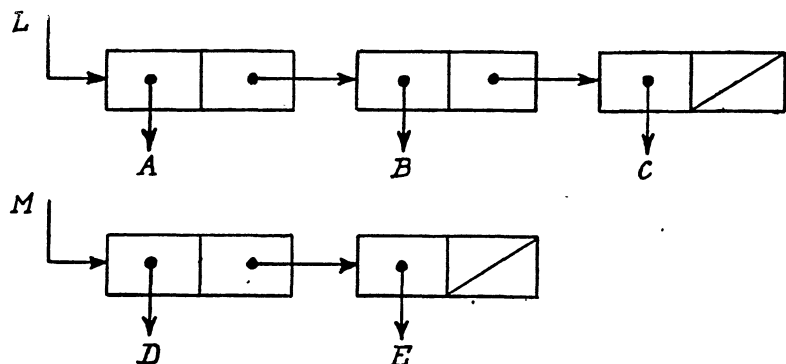
SDR : (SDR $lista_1$ $lista_2$) ha lo stesso effetto di QDR, e come valore il valore vero;
 es.
 (SDR (A B) (C D))
 TRUE

XDR : (XDR $lista_1$ $lista_2$) ha lo stesso effetto di QDR, e come valore il CDR di $lista_2$;
 es.
 (XDR (A B) (C D))
 (D)

NCONC : (NCONC $lista_1$ $lista_2$) ha come effetto la modifica fisica di $lista_1$ in quanto inserisce come CDR del suo ultimo nodo $lista_2$, e come valore la lista ottenuta concatenando $lista_2$ a $lista_1$;
 es.
 (NCONC (A B) (C D))
 (A B C D)

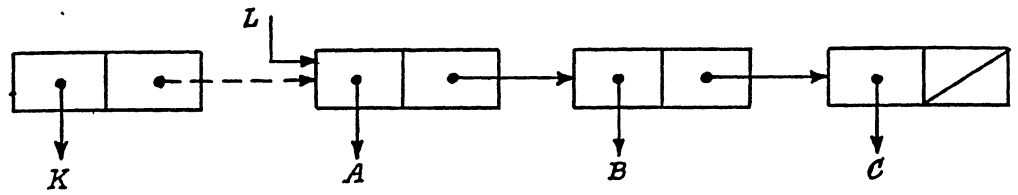
Per visualizzare il comportamento delle primitive per la manipolazione di liste è utile (anche in sede di programmazione), fornire la rappresentazione grafica delle equazioni che si effettuano.

La lista L sia (A B C) e la lista M (D E); la loro rappresentazione è:

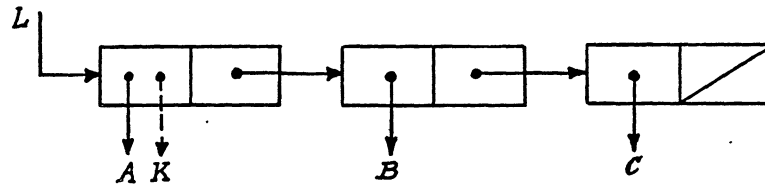


Le operazioni suddette si visualizzano come segue (le linee tratteggiate sostituiscono quelle continue):

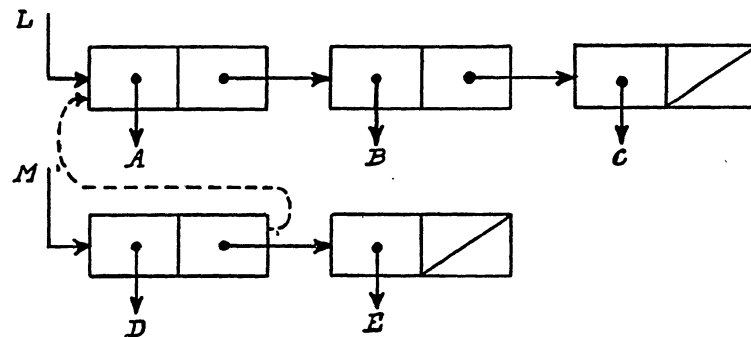
(EVAL (CONS (QUOTE K) L))



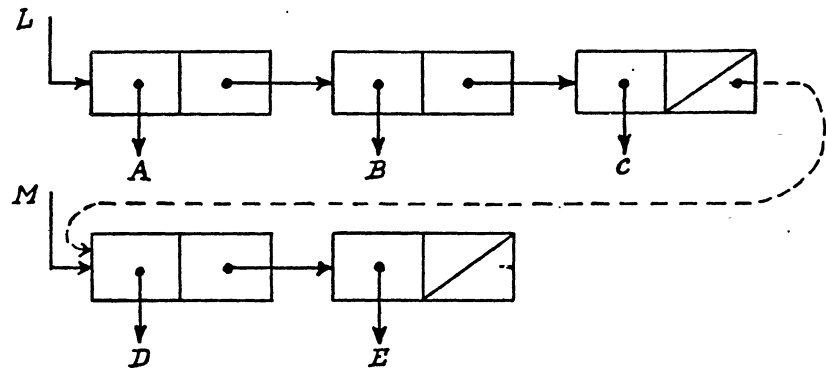
(EVAL (QAR (QUOTE K) L))



(EVAL (QDR L M))



(EVAL (NCONC L M))



3.1.3. Manipolazione di atomi

MATOM : (**MATOM** *namtab*), dove *namtab* deve essere una tavola di $n \leq 5$ righe, il cui contenuto è una stringa di m caratteri ($4(n-1) < m \leq 4n$), ha come valore l'atomo costituito dalla suddetta stringa, e come effetto la costruzione in memoria di due tavole che costituiscono la struttura dell'atomo;

BUCKET : (**BUCKET** *namtab*) ha come valore la lista del registro degli atomi nella quale, l'atomo, il cui nome è contenuto in *namtab*, è stato classificato secondo una procedura descritta in altro lavoro;

MAKEAT : (**MAKEAT** *namtab*) ha come eventuale effetto la costruzione in memoria dell'atomo, il cui nome è contenuto in *namtab*, (se non è ancora noto all'interprete, cioè se il suo nome non è ancora contenuto nel registro degli atomi), e come valore l'atomo il cui nome è contenuto in *namtab*;

VAL : (**VAL** *atomo*) ha come valore l'oggetto che costituisce il valore di *atomo*;

PNAME : (**PNAME** *atomo*) ha come valore la tavola del *printname* di *atomo*;

LETPNAME : (**LETPNAME** *atomo namtab*) ha come effetto l'assegnazione di *namtab* (contenente una stringa di caratteri) come nuovo *printname* di *atomo*, e come valore il valore vero;

LET : (**LET** *atomo obj*) ha come effetto l'assegnazione di *obj* come nuovo valore di *atomo*, e come valore il valore vero;
es. (**LET** A B)
TRUE
(**VAL** A)
B

DEFINE : (**DEFINE** *namvallist*) dove *namvallist* deve essere una lista di $n \geq 1$ liste del tipo (*atomo_i obj_i*), ha come effetto l'assegnazione di *obj_i* come nuovo valore del rispettivo *atomo_i*, e come valore la lista di tutti gli *atomo_i* ($1 \leq i \leq n$).
Es. (**DEFINE** ((A1 B1)(A2 B2)(A3 B3)))
(A1 A2 A3)
(**VAL** A1)
B1
(**VAL** A2)
B2
(**VAL** A3)
B3

3.1.4. Manipolazione di tavole

- MAKEARR: (MAKEARR *num logic₁ logic₂*) ha come effetto la costruzione di una tavola di *num* elementi ciascuno dei quali è lista vuota (se *logic₁* è il valore vero), oppure è costituito da una stringa di quattro caratteri neutri (se *logic₁* è il valore falso) a partire dalla prima parola libera (se *logic₂* è il valore falso) o dalla prima parola libera di indirizzo pari (se *logic₂* è il valore vero); ha come valore la tavola costruita;
- PUTBCD: (PUTBCD *tab₁ tab₂ num*) ha come effetto il trasferimento in *tab₂*, a partire dal *num*-esimo carattere, dei caratteri, diversi dal carattere neutro, contenuti in *tab₁*; ha come valore la somma di *num* e del numero di caratteri, diversi dal carattere neutro, contenuti in *tab₁*;
- ENT: (ENT *obj num tab*) ha come effetto l'inserimento di *obj* come *num*-esimo elemento di *tab*, ha come valore il valore vero;
- EXT: (EXT *num tab*) ha come valore il *num*-esimo elemento di *tab*;
- ENTCH: (ENTCH *onecaratom num tab*) dove *onecaratom* è un atomo il cui nome è costituito da un solo carattere, ha come effetto l'inserimento del carattere che costituisce il printname di *onecaratom* come *num*-esimo carattere di *tab* e come valore il valore vero;
- EXTCH: (EXTCH *num tab*) ha come valore l'atomo il cui printname è costituito dal *num*-esimo carattere di *tab*.

3.1.5. Operazioni sui numeri interi

- INCR: (INCR *num*) ha come valore *num+1*
- DECR: (DECR *num*) ha come valore *num-1*
- ADD: (ADD *num₁ num₂*) ha come valore *num₁+num₂*
- SUB: (SUB *num₁ num₂*) ha come valore *num₁-num₂*
- MULT: (MULT *num₁ num₂*) ha come valore *num₁·num₂*
- DIV: (DIV *num₁ num₂*) ha come valore $\lfloor \text{num}_1 / \text{num}_2 \rfloor$
- ABS: (ABS *num*) ha come valore $|\text{num}|$
- NEG: (NEG *num*) ha come valore $-\text{num}$
- REM: (REM *num₁ num₂*) ha come valore $\text{num}_1 - \left\lfloor \frac{\text{num}_1}{\text{num}_2} \right\rfloor \cdot \text{num}_2$

N.B. Nessuna segnalazione viene fornita se si supera il massimo numero permesso dall'interprete ($2^{21}-1$).

3.1.6. Funzionali

MAP (MAP *lista function*) dove *function* è una funzione LISP, ha come effetto la valutazione delle forme ottenute applicando *function* a *lista* e ai suoi CDR successivi fino a che si ottiene il valore falso oppure si esaurisce *lista*; nel primo caso il valore è la lista che ha dato luogo al valore falso, nel secondo il valore è la lista vuota;

APPLY : (APPLY *function lista*) ha come valore quello che deriva dalla valutazione della forma ottenuta applicando *function* agli elementi di *lista*;

EVAL (EVAL *dato*) ha l'effetto di valutare *dato* ed ha come valore il valore di *dato*.

Es. (MAP (A B C) (LAMBDA (X) (LET (CAR X) ())))

()

(VAL A)

()

(VAL B)

()

(VAL C)

()

(APPLY CONS (A(B C)))

(A B C)

(CONS A(B C))

(A B C)

(EVAL (CONS (QUOTE A) (QUOTE(B C))))

(A B C)

3.1.7. Generazione di simboli

LPAREN: (LPAREN) ha come valore lo pseudoatomo costituito dal carattere (;

RPAREN: (RPAREN) ha come valore lo pseudoatomo costituito dal carattere);

BLANK : (BLANK) ha come valore lo pseudoatomo costituito dal carattere b;

COMMA : (COMMA) ha come valore lo pseudoatomo costituito dal carattere ,;

ZERZER: (ZERZER) ha come valore lo pseudoatomo costituito dal carattere.

3.1.8. Operazioni di I/O

- NEXT : (NEXT) se l'indice di lettura del buffer di entrata (di 72 caratteri) è proporzionato in corrispondenza del K -esimo carattere ($1 \leq K < 72$), esso viene spostato sul carattere $K+1$, se è posizionato sul 72° carattere, l'effetto di NEXT è di ricaricare il buffer (con una operazione di entrata) e di riportare l'indice sul primo carattere. Ha come valore l'atomo di un solo carattere costituito dal carattere su cui è puntato l'indice dopo lo spostamento.
- NEXT* : (NEXT*) ha come valore l'atomo costituito dal carattere successivo a quello su cui è puntato l'indice di lettura del buffer di entrata.
- WBCD : (WBCD *namtab*) dove *namtab* deve essere una tavola contenente una stringa di m caratteri diversi dal carattere neutro. Se l'indice di scrittura del buffer di uscita (120 caratteri) è puntato sulla K -esima posizione⁽¹⁾ la stringa contenuta in *namtab* viene trascritta nel buffer di uscita a partire dalla detta posizione e l'indice viene quindi spostato sulla posizione $K+m$: se $K+m$ è maggiore di 120, prima di effettuare la trascrizione il buffer viene scaricato con una operazione di stampa, quindi viene riempito di caratteri spazio, la stringa contenuta in *namtab* viene trascritta a partire dalla prima posizione del buffer e l'indice spostato sulla posizione m -esima. Ha come valore il valore vero.
- SPACE : (SPACE) ha come effetto lo spostamento di una posizione in avanti dell'indice del buffer di uscita (con lo stesso comportamento di WBCD) e come valore il valore vero.
- WRITE : (WRITE) ha l'effetto di scaricare, con una operazione di stampa, il contenuto del buffer di uscita, quindi riempie il buffer di caratteri spazio e riporta l'indice sulla K -esima posizione. Ha come valore il valore vero.

(1) Posizione è la zona di memoria che contiene esattamente un carattere.

- DEC : (DEC *namtab*) se il primo carattere della stringa di caratteri alfanumerici contenuta nella tavola *namtab* è un carattere numerico o il carattere "-" e se tutti gli altri caratteri diversi da quello neutro sono numerici, ha come valore il numero stesso mod 2^{21} , in caso contrario ha come valore la stessa tavola *namtab*.
- UNDEC : (UNDEC *num*) ha come valore la tavola di due righe contenente la stringa di caratteri alfanumerici che rappresentano *num* (eventualmente preceduta dal segno "-") seguita da carattere neutro fino ad esaurimento delle due righe.
- READIN : (READIN) ha l'effetto di esaurire il contenuto del buffer di entrata (e nel caso che esso sia vuoto lo carica con una operazione di entrata da fotolettore o da telescrivente secondo la posizione di una chiave di console) e di costruire in memoria la struttura del dato LISP ivi letto. Ha come valore il dato stesso.
- PRINT : (PRINT *outdato*) dove *outdato* deve essere un numero, un atomo, una lista o un valore di verità. Ha l'effetto di eseguire una operazione di uscita (su stampante o su telescrivente secondo la posizione di una chiave di console) e ha come valore *outdato*.
- WMAG : (WMAG *unit magtab*) dove $1 \leq \text{unit} \leq 4$ e *magtab* è una tavola posta ad indirizzo pari e dotata di un numero pari di righe. Ha l'effetto di registrare sul nastro magnetico posto sull'unità *unit* il blocco costituito dalla stringa contenuta in *magtab* e, se la scrittura è avvenuta correttamente, il valore è il valore vero, se vi è stato il rilevamento di un qualunque errore durante l'operazione con l'unità, il valore è il valore falso.
- RMAG : (RMAG *unit magtab*) ha l'effetto di leggere dall'unità *unit* un blocco di caratteri che vengono inseriti nella tavola *magtab* a partire dalla prima posizione fino ad esaurimento di *magtab* e, se la lettura è avvenuta correttamente il valore è il numero di caratteri letti, se invece vi è stato il rilevamento di un qualunque errore durante l'operazione con l'unità il valore è il valore falso.
- CMAG : (CMAG *unit op*) dove $0 \leq \text{op} \leq 6$ rappresenta l'operazione da eseguire sull'unità *unit*. Ha i seguenti effetti:
- op* = 0 riavvolgimento
 - op* = 1 spost. in avanti fino a dopo il primo blocco unitario

$op = 2$ spost. indietro fino a dopo il primo blocco unitario
 $op = 3$ spost. indietro di un blocco
 $op = 4$ spost. in avanti fino a dopo la prima marca
 $op = 5$ cancellazione di lunghezza fissa (~ 10 cm)
 $op = 6$ cancellazione totale.

Se l'operazione è effettuata correttamente il valore è il valore vero, se vi è stato il rilevamento di un qualunque errore, il valore è il valore falso.

UMAG (UMAG word) dove $0 \leq word \leq 4095_{10}$ ($= 7777_8$). Se $word = Y_1Y_2Y_3Y_4_{10}$ ($= X_1X_2X_3X_4_8$) UMAG ha l'effetto di registrare il blocco unitario costituito dai due caratteri A_1A_2 (alfanumerici) la cui codificazione ottale è, rispettivamente, X_1X_2 e X_3X_4 , sul nastro posto sull'unità unit.

Se l'operazione è effettuata correttamente il valore è il valore vero, se vi è stato il rilevamento di un qualunque errore, il valore è il valore falso.

EMAG : (EMAG unit num) ha l'effetto di cancellare il nastro posto sull'unità unit per una lunghezza pari a $4 \times num$ caratteri e ha come valore il valore vero se l'operazione è effettuata correttamente, il valore falso se vi è stato il rilevamento di un qualunque errore.

3.1.9. Operazioni di servizio

RESTART: (RESTART) ha l'effetto di reinizializzare l'interprete.

ERROR : (ERROR lista₁) ha l'effetto di stampare una lista il cui CAR è l'atomo ERROR ed il cui CDR è lista₁ e di reinizializzare l'interprete.

GC : (GC obj) ha l'effetto di eseguire una "garbage collection" e ha come valore obj.

PAUSE : (PAUSE) ha l'effetto di porre l'interprete ROMALISP in stato d'attesa e ne invia notizia all'operatore con il messaggio PAUSE. Per riavviare l'elaborazione si deve premere il tasto VIA sulla consolle operativa. Ha come valore il valore vero.

STOP : (STOP) ha l'effetto di arrestare completamente l'elaborazione e ne invia notizia all'operatore con il messaggio STOP. A questo punto è possibile solo un rientro al sistema operativo mediante operazioni da consolle (tasti SG, VIA).

OPERATE: (OPERATE) ha l'effetto di passare il controllo al sistema operativo e ne invia notizia all'operatore con il messaggio OPERATE. L'entrata nel sistema operativo viene segnalata dal messaggio READY OP.

3.2. FUNZIONI SPECIALI

Con il nome di funzioni speciali indichiamo alcuni nomi chiave che determinano un particolare meccanismo di valutazione, e cioè:

1. il numero di argomenti è variabile,
2. non tutti gli argomenti vengono necessariamente valutati (come invece accade per le funzioni descritte in 3.1),
3. non possono essere valutate attraverso APPLY perché la valutazione degli argomenti è indispensabile al corretto funzionamento (tranne in alcuni casi particolari).

Per tali motivi i modelli di forma funzionale indicheranno l'esistenza degli argomenti ma non la classe del loro valore e la definizione non indicherà semplicemente l'effetto e il valore della applicazione della funzione ai valori degli argomenti ma anche il metodo di valutazione degli argomenti stessi.

QUOTE : (QUOTE *arg*)
arg non viene valutato; il valore della forma è *arg*.

SETQ : (SETQ *arg*₁ *arg*₂)
*arg*₁ deve essere un atomo e non viene valutato;
*arg*₂ viene valutato e il suo valore (un qualunque oggetto viene assegnato come valore di *arg*₁ ed è il valore della forma.

Esempi: il valore di L sia la lista (A B C)

(EVAL (CAR (QUOTE (A B C))))

A

(CAR (QUOTE (A B C)))

QUOTE

(EVAL (LET (QUOTE A) L))

TRUE

(VAL A)

(A B C)

(EVAL (SETQ A (CDR A)))

(B C)

(VAL A)

(B C)

LIST : (LIST $arg_1 arg_2 \dots arg_n$) $n \geq 0$
 ha come valore la lista che ha per elementi i valori di $arg_1, arg_2, \dots, arg_n$ (eventualmente la lista vuota).
 Esempio: il valore di L sia la lista (A B C)
 (LIST D)
 (D)
 (EVAL (LIST (CAR L)(CADR L)(CADDR L)))
 (A B C)

OR : (OR $arg_1 \dots arg_n$) $n \geq 0$
 gli argomenti vengono valutati fino a che non se ne incontra uno il cui valore sia il valore vero (o un numero positivo o nullo): se tale argomento esiste il suo valore è il valore della forma, se non esiste (e quindi in particolare se $n = 0$) il valore della forma è il valore falso.

AND : (AND $arg_1 \dots arg_n$) $n \geq 0$
 gli argomenti vengono valutati fino a che non se ne incontra uno il cui valore sia il valore falso (o un numero negativo o un qualunque altro oggetto): se tale argomento esiste il suo valore è il valore della forma, se non esiste (e quindi in particolare se $n = 0$) il valore della forma è il valore vero.
 Esempi: sia la lista (A B C) il valore di L ed E il valore di F
 (OR)
 FALSE
 (AND)
 TRUE
 (EVAL (AND (LIS L)(LIS (CDR L))(LIS (CDDR L))))
 TRUE
 (EVAL (OR (ATOM L)(NUM L)(LIS (CAR L))))
 FALSE
 (EVAL (OR (ATOM L)(LET L F)(LET L (QUOTE (A B C)))))
 TRUE
 (VAL L)
 E

IF : (IF $arg_1 arg_2 arg_3$)
 se il valore di arg_1 è il valore vero viene valutato arg_2 e il suo valore è il valore della forma, se il valore di arg_1 è il valore falso viene valutato arg_3 e il suo valore è il valore della forma.

COND : (COND ($arg_{11} arg_{12}$) ... ($arg_{n1} arg_{n2}$)) $n \geq 0$
 gli arg_{i1} vengono valutati fino a che non se ne incontra uno arg_{k1} , il cui valore è il valore vero: al-

lora viene valutato il corrispondente arg_{k_2} e il suo valore è il valore della forma; se tale argomento non esiste (in particolare, quindi, se $n = 0$) il valore della forma è il valore falso.

Esempi: sia A_i il valore di A_i

```
(EVAL (SETQ SW (TRUE)))
```

TRUE

```
(EVAL (IF SW A1 A2))
```

A1

```
(EVAL (SETQ SW (FALSE)))
```

FALSE

```
(EVAL (IF SW A1 A2))
```

A2

```
(EVAL (COND (SW A1)(SW A2)((NOT SW)A3)(SW A4)))
```

A3

COMMENT: (COMMENT $arg_1 \dots arg_n$) $n \geq 0$

questa forma ha la funzione di inserire commenti fra un programma e l'altro: non viene valutata in alcun modo e non deve essere inclusa in un programma.

Esempio:

```
(COMMENT: IAC - CNR (DICEMBRE 1968))
```

3.3. FUNZIONI LAMBDA-DEFINITE

******La definizione di funzioni ricorsive in LISP viene effettuata mediante la notazione λ (introdotta dal logico A.Church) che consente di distinguere una funzione da una forma funzionale.

L'espressione $e = 2x + 3y + 4$ ha infatti, nel linguaggio comune, il significato di legge che a ogni coppia di valori di x e di y associa il valore di e ma, contemporaneamente, fissati i valori di x e y (per esempio, rispettivamente 1 e 5) ha il significato di valore di e (per es. 21); il primo significato è quello proprio di una funzione, il secondo è quello proprio di una forma funzionale: la notazione lambda ha il compito di evitare tale ambiguità: $2x + 3y + 4$ rappresenta allora la forma funzionale, $(\lambda(xy)(2x+3y+4))$ la funzione che a due valori di x e y (es. $\bar{x}$ e $\bar{y}$) associa il valore $2\bar{x} + 3\bar{y} + 4$.

La distinzione introdotta comporta anche una distinzione nelle variabili: le variabili che compaiono nella lista che segue la λ prendono il nome di variabili legate e non sono altro che simboli utilizzati per definire le operazioni da effettuare sugli argomenti. Per contrasto, tutte le altre variabili

prendono il nome di variabili libere.

Il LISP riprende la medesima notazione per consentire la definizione di funzioni ottenute per composizione di funzioni primitive, compilate o speciali.

Esempio: il predicato ATOMIC è vero per atomi e numeri:

```
(DEFINE ((ATOMIC (LAMBDA (X) (OR (ATOM X)
                                   (NUM X) )))
        ))
(ATOMIC)
(ATOMIC (A B C))
FALSE
```

La particolare necessità di usare la notazione LAMBDA deriva dalla necessità di definire funzioni ricorsive, cioè funzioni nella cui definizione viene utilizzata la funzione stessa.

Esempio: se M ed N sono due liste APPEN* concatena la lista N ad una copia della lista M

```
(DEFINE (
  (APPEN* (LAMBDA (M N) (COND
    ((NULL M) N)
    ((NULL N) M)
    ((TRUE) (CONS (CAR M)
                   (APPEN* (CDR M) N))))))
  ))
(APPEN*)
(APPEN* (A B C) (D E F))
(A B C D E F)
```

**

Il modello di forma funzionale in cui il primo elemento è una lambda definizione è il seguente:

$$((\text{LAMBDA } (var_1 \dots var_n) \text{ dato}) arg_1 \dots arg_m) \quad \begin{matrix} n \geq 0 \\ m \geq 0 \end{matrix}$$

dove $var_1, \dots, var_n$ sono atomi e costituiscono le variabili legate e *dato* è un numero, un atomo o una lista in cui (generalmente) occorrono le variabili legate.

La valutazione della forma suddetta conviene nel seguente modo:

- vengono valutati gli m argomenti,
- i valori delle variabili legate vengono memorizzati,
- il valore dei primi n argomenti viene assegnato come (nuovo) valore alle prime m variabili legate e se $n < m$ le rimanenti $n-m$ variabili legate assumono come valore la lista vuota,
- viene valutato *dato* (utilizzando così, implicitamente, i valori delle variabili legate),

- i vecchi valori delle variabili legate vengono ad esse restituiti cancellando quelli assunti temporaneamente,
- il valore di *dato* (ottenuto precedentemente) viene assunto come valore della forma.

3.4. FUNZIONI RICORSIVE E FUNZIONI ITERATIVE

******La lambda definizione delle funzioni consente, come è stato detto, di introdurre funzioni definite mediante se stesse, cioè tali che la valutazione delle relative forme funzionali implica l'applicazione delle funzioni stesse a nuovi argomenti. La necessità di memorizzare tutti gli argomenti ai quali la funzione è stata ricorsivamente applicata comporta un'organizzazione a stack dei valori successivamente assunti dalle variabili legate della lambda definizione e un meccanismo ricorsivo anche nella valutazione delle stesse forme funzionali. Gli aspetti negativi dell'impiego di funzioni definite ricorsivamente sono essenzialmente due:

- il possibile esaurimento dello stack dell'interprete nel caso che i passi di ricursione siano troppo numerosi,
- l'inutile ingombro di memoria costituito da argomenti di funzioni ricorsive la cui memorizzazione non è necessaria.

Esempio: Se *L* è una lista *LAST* ne fornisce l'ultimo elemento:

```
(DEFINE (
  (LAST (LAMBDA (L) (IF (NULL (CDR L))
                        (CAR L)
                        (LAST (CDR L)) )))
  ))
(LAST)
(LAST (A B C))
C
```

Come si vede, ad ogni chiamata ricorsiva di *LAST* il precedente argomento, cioè il precedente valore di *L*, può essere distrutto. In questi casi il LISP consente la definizione iterativa di funzioni per mezzo della struttura a programma ("program feature"); essa consiste in una serie di forme funzionali delle quali non interessa il valore ma l'effetto della valutazione e che quindi si configurano come istruzioni di un programma. Le caratteristiche della "program feature", che consentiva l'effettuazione di cicli senza chiamate ricorsive, sono:

- l'assegnazione di nomi simbolici a sequenza di istruzioni,

- l'uso di istruzioni di salto (GO α)
- l'uso di istruzioni di uscita della struttura a programma (RETURN β)
- l'uso di variabili proprie delle quali non è effettuata, all'interno della struttura a programma, alcuna memorizzazione. **

Il modello di forma funzionale a struttura di programma è seguente:

$$(\text{PROG } (var_1 \dots var_n) arg_1 \dots arg_m) \quad \begin{array}{l} n \geq 0 \\ m > 0 \end{array}$$

dove $var_1, \dots, var_n$ sono atomi e rappresentano le variabili proprie della "program feature" e $arg_1, \dots, arg_m$ sono liste o atomi o numeri.

La valutazione della forma suddetta avviene nel seguente modo:

- i valori delle variabili proprie vengono memorizzati,
- alle variabili proprie viene assegnato come valore lista vuota,
- $arg_1, \dots, arg_m$ vengono esaminati con il seguente criterio:
 - se arg_i è un atomo o un numero viene considerato un'etichetta di salto e quindi non viene valutato e si esamina arg_{i+1} ;
 - se arg_i è una lista viene considerata una forma funzionale e viene valutata. Tale valutazione avviene sempre tramite la funzione EVAL e valgono quindi per essa le regole viste in 2.4 e in 3. ma oltre alle forme funzionali del tipo visto in 2.4 e in 3., all'interno di una "program feature" si possono avere forme del tipo:

(GO arg) la cui valutazione avviene nel seguente modo:
se arg è una lista viene valutata e il suo valore deve essere un atomo o un numero che siano stati usati all'interno della PROG come etichetta, se arg è un atomo o un numero non viene valutato: in entrambi i casi viene ricercato fra gli arg_i della forma PROG l'atomo o il numero corrispondente e quindi la valutazione degli arg_i viene proseguita da quello successivo all'etichetta di salto così individuata. Se tale etichetta non viene trovata o essa è l'ultimo arg della lista si ha una segnalazione d'errore.

(RETURN arg) la cui valutazione avviene nel seguente modo:
 arg viene valutato, la valutazione della for-

ma PROG viene interrotta, come suo valore si assume proprio il valore di *arg* e alle variabili viene restituito il valore che avevano precedentemente.

In ogni caso, il valore di un *arg_i* (la cui valutazione non abbia determinato una operazione RETURN) viene ignorato e si esamina *arg_{i+1}*;

• se l'*arg_i* valutato era l'ultimo della forma PROG, alle variabili proprie viene restituito il valore precedente e il valore della forma PROG è la lista vuota.

Esempi: la funzione LAST è quella definita precedentemente, la funzione FACT è il fattoriale del numero R

```
(DEFINE (
  (LAST (LAMBDA (L) (PROG ( )
    ALF (COND ((NULL (CDR L))(RETURN (CAR L))))
    (SETQ L (CDR L))
    (GO ALF) )))
  (FACT (LAMBDA (R) (PROG (N)
    (SETQ N 1)
    111 (COND ((ZERO R)(RETURN N)))
    (SETQ N (MULT R N))
    (SETQ R (DECR R))
    (GO 111) )))
  ))
(LAST FACT)
(LAST (A B C))
C
(FACT 5)
120
```

APPENDICI

A. - DEFINIZIONE LISP DELL' INTERPRETE

```
{PROCESS (LAMBDA ()
  (AND (TRUTH ((LAMBDA (X)
    (PRINT (IF (OR (NULL X)
      (NOT (LIS X)))
      X
      (IF (EQ (CAR X)
        (QUOTE COMMENT))
        ((LAMBDA (V)
          (BLANK))
          (SETQ FREELIST (NCONC X FREELIST)))
          (APPLY (CAR X)
            (CDR X))))))
    (PRINT (READIN))))
  (RESTART))))
(TRUTH (LAMBDA (X)
  (TRUE)))
(READIN (LAMBDA ()
  ((LAMBDA (C)
    (COND
      ((EQ C (BLANK))
        (READIN))
      ((EQ C (LPAREN))
        (READLIST))
      ((EQ C (RPAREN))
        (TRUE))
      ((TRUE)
        (READATOM))))
    (NEXT))))
(READLIST (LAMBDA ()
  ((LAMBDA (X)
    (IF (TRUVAL X)
      (LIST)
      (CONS X (READLIST))))
    (READIN))))
(READATOM (LAMBDA ()
  ((LAMBDA (R)
    ((LAMBDA (A)
      (IF (NUM A)
```

```

A.
(MAKEATOM (BUCKET R))))
(DEC R)))
(PRINTN 1))))
(PRINTN (LAMBDA (N)
((LAMBDA (CC)
(IF (OR (EQ CC (BLANK))
(EQ CC (LPAREN))
(EQ CC (RPAREN))))
((LAMBDA (NN)
((LAMBDA (R)
((LAMBDA (Y)
((LAMBDA (Z)
R)
(ENTCH C N R))))
(ENT 10808 NN R)))
(EXT NN PNR)))
(DIV (ADD N 3)
4))
((LAMBDA (W)
((LAMBDA (C)
((LAMBDA (Y)
((LAMBDA (Z)
Y)
(ENTCH W N Y))))
(PRINTN (INCR N))))
(NEXT)))
C)))
(NEXT*))))
(MAKEATOM (LAMBDA (L)
(COND
((ARREQ R (PNAME (CAR L)))
(CAR L))
((NULL (CDR L))
((LAMBDA (A)
(CAR (RDR (LIST A)
L))))
(MATOM R)))
((AND)
(MAKEATOM (CDR L))))))
(PRINT (LAMBDA (X)
((LAMBDA (Z)
X)
(AND (WRITOBJECT X)
(WRITE))))))

```

```
(WRITOBJECT (LAMBDA (X)
  (IF (LIS X)
    (AND (WBCD (PNAME (LPAREN)))
      (WRITEFRAGMENT X)
      (WBCD (PNAME (RPAREN)))))
    (AND (WBCD (COND
      ((ATOM X)
        (PNAME X))
      ((NUM X)
        (UNDEC X))
      ((TRUVAL X)
        (PNAME (IF X (QUOTE TRUE)
          (QUOTE FALSE)))))
      ((TRUE)
        (PNAME (QUOTE ****)))))))
(WRITEFRAGMENT (LAMBDA (X)
  (OR (NULL X)
    (AND (TRUTH (AND (WRITOBJECT (CAR X))
      (AND (NOT (LIS (CAR X)))
        (NOT (NULL (CDR X))),
        (SPACE))))
      (WRITEFRAGMENT (CDR X))))))
(APPLY (LAMBDA (X L)
  (EVAL (CONS X (APPQ L)))))
(APPQ (LAMBDA (L)
  (IF (NULL L)
    L
    (CONS (LIST (QUOTE QUOTE)
      (CAR L))
      (APPQ (CDR L)))))
(EVAL (LAMBDA (E)
  (PROG (F A)
    EVL
    (COND
      ((ATOM E)
        (RETURN (VAL E)))
      ((OR (NUM E)
        (NULL E)
        (TRUVAL E))
        (RETURN E)))
    (SETQ F (CAR E))
    (SETQ A (CDR E))
    APP
    (COND
      ((ATOM F)
        (GO EA))
```

```
((NOT (LIS F))
 (ERROR (LIST 1 F)))
(EQ (CAR F)
 (QUOTE LAMBDA))
(GO EL)))
(SETQ F (EVAL F))
(GO APP)
EA
(COND
 ((PRIM (VAL F))
  (RETURN (CALLP (VAL F)
                 (EVAR A 1)))))
 ((EQ F (QUOTE AND))
  (GO EVAND))
 ((EQ F (QUOTE OR))
  (GO EVOR))
 ((EQ F (QUOTE IF))
  (GO EVIF))
 ((EQ F (QUOTE COND))
  (GO EVCOND))
 ((EQ F (QUOTE SETQ))
  (GO EVSETQ))
 ((EQ F (QUOTE LIST))
  (GO EVLIST))
 ((EQ F (QUOTE QUOTE))
  (RETURN (CAR A)))
 ((EQ F (QUOTE GO))
  (GO EVGO))
 ((EQ F (QUOTE RETURN))
  (GO EVRETURN))
 ((EQ F (QUOTE PROG))
  (GO EVPROG))
 ((CSUBR (VAL F))
  (RETURN (CALLC 4 (VAL F)
                   (EVAR A 1)))))
 ((EQ F (VAL F))
  (ERROR (LIST 2 F)))))
(SETQ F (VAL F))
(GO APP)
EVIF
(SETQ E (COND
 ((EVAL (CAR A))
  (CADR A))
 ((TRUE)
  (CADDR A)))))
(GO EVL)
```

```
EVCOND
(COND
  ((NULL A)
   (RETURN (FALSE)))
  ((EVAL (CAAR A))
   (GO EC1)))
(SETQ A (CDR A))
(GO EVCOND)
EC1
(SETQ E (CADAR A))
(GO EVL)
EVAND
(COND
  ((NULL A)
   (RETURN (TRUE)))
  ((NOT (EVAL (CAR A)))
   (RETURN (FALSE))))
(SETQ A (CDR A))
(GO EVAND)
EVOR
(COND
  ((NULL A)
   (RETURN (FALSE)))
  ((EVAL (CAR A))
   (RETURN (TRUE))))
(SETQ A (CDR A))
(GO EVOR)
EL
(SETQ A (XVAL (EVAR A 1)
              (CADR F)))
(SETQ E (EVAL (CADDR F)))
(RVAL (CADR F)
      A)
(RETURN E)
EVLIST
(COND
  ((NULL A)
   (RETURN ())))
(SETQ E (SETQ F (LIST (EVAL (CAR A)))))
ET1
(COND
  ((NULL (SETQ A (CDR A)))
   (RETURN F)))
(SETQ E (RDR (LIST (EVAL (CAR A))
                  E)))
(GO ET1)
```

```
EVSETQ
(COND
  ((NOT (ATOM (CAR A)))
   (ERROR (LIST 3 (CAR A)))))
(RETURN ((LAMBDA (X)
          ((LAMBDA (Y)
            X)
           (LET (CAR A)
               X)))
         (EVAL (CADR A)))))
EVRETURN
(SETQ PROGV (EVAL (CAR A)))
(SETQ PROGC ())
(RETURN (TRUE))
EVGO
(SETQ A (COND
  ((LIS (CAR A))
   (EVAL (CAR A)))
  ((TRUE)
   (CAR A))))
(SETQ E PROGL)
EG1
(COND
  ((NULL (SETQ E (CDR E)))
   (ERROR (LIST 4 A)))
  ((NOT (EQ A (CAR E)))
   (GO EG1)))
(SETQ PROGC E)
(RETURN (TRUE))
EVPROG
(RETURN (PROG (PROGL PROGC PROGV)
  (SETQ PROGV ())
  (SETQ PROGL (SETQ PROGC A))
  (SETQ F (XVAL 1 (CAR A)))
  EP3
  (COND
    ((NULL (SETQ PROGC (CDR PROGC)))
     (GO EP4))
    ((NOT (LIS (CAR PROGC)))
     (GO EP3)))
  (EVAL (CAR PROGC))
  (GO EP3)
  EP4
  (RVAL (CAR A)
   F)
  (RETURN PROGV))))))
```

```
(XVAL (LAMBDA (X V)
  (IF (NULL V)
    ()
    (PROG (Y)
      (SETQ Y 1)
      NV
      (COND
        ((NULL V)
          (GO CV))
        ((LE X Y)
          (ENT ()
            Y
            AR)))
        ((LAMBDA (T)
          ((LAMBDA (H)
            (ENT T Y AR))
            (LET (CAR V)
              (EXT Y AR))))
          (VAL (CAR V)))
        (SETQ V (CDR V))
        (SETQ Y (INCR Y))
        (GO NV)
        CV
        (COND
          ((EQ Y 1)
            (RETURN V)))
          (SETQ Y (DECR Y))
          (SETQ V (CONS (EXT Y AR)
            V))
          (GO CV))))))
(RVAL (LAMBDA (X V)
  (IF (NULL X)
    (TRUE)
    (PROG (A)
      (SETQ A V)
      NR
      (LET (CAR X)
        (CAR V))
      (COND
        ((NULL (CDR X))
          (GO FR))
        (SETQ X (CDR X))
        (SETQ V (CDR V))
        (GO NR)
        FR
        (XDR FREELIST V)
```

```
(SETQ FREELIST A)
(RETURN (TRUE))))))
(EVAR (LAMBDA (A N)
  (IF (NULL A)
    0
    ((LAMBDA (Y)
      ((LAMBDA (W)
        ((LAMBDA (Z)
          (INCR W))
          (ENT Y N AR)))
        (EVAR (CDR A)
          (INCR N))))
        (EVAL (CAR A))))))
(DEFINE (LAMBDA (L)
  (IF (NULL L)
    L
    (COND
      ((LET (CAAR L)
        (CADAR L))
        (CONS (CAAR L)
          (DEFINE (CDR L)))))))
(MAP (LAMBDA (L F)
  (IF (NULL L)
    L
    ((LAMBDA (E)
      (IF (AND (TRUVAL E)
        (NOT E))
        L
        (MAP (CDR L)
          F)))
      (F L))))))
(MAKEAT (LAMBDA (R)
  (MAKEATOM (BUCKET R))))
```

B. - BASYC-SYSTEM

```
(GENCNT 0)

(MAPLIST (LAMBDA ($L $F)
  ((LAMBDA ($V)
    (PROG2 (MAP $L (QUOTE (LAMBDA ($J)
      (LET (QUOTE $V)
        (CONS ($F $J)
          $V))))))
    (DREVERSE $V)))
  ())))

(PROG1 (LAMBDA ($X $Y)
  $X))

(PROG2 (LAMBDA ($X $Y)
  $Y))

(GENSYM (LAMBDA ()
  (PACK (LIST (QUOTE *)
    (PROG2 (LET (QUOTE GENCNT)
      (INCR GENCNT))
      GENCNT))))))

(PACK (LAMBDA ($L)
  ((LAMBDA ($R)
    (PACK1 $L 1))
    (MAKEARR (DIV (MAPRED $L (QUOTE (LAMBDA ($J $K)
      (ADD ((LAMBDA ($N)
        (DECR (PUTBCD $N $N 1)))
        (NNAME $J))
      $K)))
    3)
    4)
    (FALSE)
    (FALSE))))))

(NNAME (LAMBDA ($A)
  (IF (ATOM $A)
    (PNAME $A)
    (UNDEC $A))))

(PACK1 (LAMBDA ($L $N)
  (IF (NULL $L)
    (MAKEAT $R)
    (PACK1 (CDR $L)
      (PUTBCD (NNAME (CAR $L))
```

```

$R
$N))))
(ASSOC (LAMBDA ($X $L)
  (CAR (MAP $L (QUOTE (LAMBDA ($J)
    (NOT (EQ $X (CAAR $J))))))))))
(MAPC (LAMBDA ($LM $FM)
  (MAP $LM (QUOTE (LAMBDA ($JM)
    ($FM (CAR $JM)))))))
(LENGTH (LAMBDA ($L)
  ((LAMBDA ($N)
    (PROG2 (MAP $L (QUOTE (LAMBDA ($J)
      (LET (QUOTE $N)
        (INCR $N))))))
    $N))
  0)))
(LAST (LAMBDA ($L)
  (MAP $L (QUOTE (LAMBDA ($J)
    (NOT (NULL (CDR $L))))))))
(EQUAL (LAMBDA ($X $Y)
  (OR (EQ $X $Y)
    (AND (LIS $X)
      (LIS $Y)
      (EQUAL (CAR $X)
        (CAR $Y))
      (EQUAL (CDR $X)
        (CDR $Y))))))
(MEMB (LAMBDA ($A $L)
  (NOT (NULL (MAP $L (QUOTE (LAMBDA ($J)
    (NOT (EQUAL $A (CAR $J))))))))))
(DREVERSE (LAMBDA ($L)
  (IF (NULL (CDR $L))
    $L
    ((LAMBDA ($P $R)
      (PROG2 (MAP (CDR $L)
        (QUOTE (LAMBDA ($J)
          (AND (LET (QUOTE $R)
            (QDR $R $P))
            (LET (QUOTE $P)
              $J))))))
        (QDR $R $P)))
    $L
    ())))))

```

```
(SUBST (LAMBDA ($X $Y $Z)
  (SUBA $Z)))
```

```
(SUBA (LAMBDA ($Z)
  (COND
    ((EQUAL $Z $Y)
     $X)
    ((NULL $Z)
     $Z)
    ((LIS $Z)
     (CONS (SUBA (CAR $Z))
            (SUBA (CDR $Z)))))
    ((TRUE)
     $Z))))
```

```
(MAPCAR (LAMBDA ($LC $FC)
  ((LAMBDA ($VC)
    (PROG2 (MAP $LC (QUOTE (LAMBDA ($JC)
      (LET (QUOTE $VC)
        (CONS ($FC (CAR $JC))
              $VC))))))
    (DREVERSE $VC)))
  ())))
```

```
(SUBPAIR (LAMBDA ($U $V $L)
  (SUBP1 $L)))
```

```
(SUBP1 (LAMBDA ($L)
  (IF (LIS $L)
    (MAPCAR $L SUBP1)
    (SUBP2 $U $V))))
```

```
(SUBP2 (LAMBDA ($U $V)
  (COND
    ((NULL $V)
     $L)
    ((EQ $L (CAR $V))
     (CAR $U))
    ((TRUE)
     (SUBP2 (CDR $U)
            (CDR $V))))))
```

```
(CHARAC (LAMBDA ($C)
  (PROG2 (ENT $C 1 CHBUF)
    (EXTCH 4 CHBUF))))
```

```
(CHCON (LAMBDA ($N)
  (PROG2 (ENT 0 1 CHBUF)
    (PROG2 (ENTCH $N 4 CHBUF)
      (EXT 1 CHBUF))))))

(MAPRED (LAMBDA ($LE $FE $VE)
  (PROG2 (MAPC (REVERSE $LE)
    (QUOTE (LAMBDA ($JE)
      (LET (QUOTE $VE)
        ($FE $JE $VE))))))
  $VE)))

(REVERSE (LAMBDA ($L)
  ((LAMBDA ($W)
    (PROG2 (MAP $L (QUOTE (LAMBDA ($J)
      (LET (QUOTE $W)
        (CONS (CAR $J)
          $W))))))
    $W)))
  ())))

(NTH (LAMBDA ($L $N)
  (MAP $L (QUOTE (LAMBDA ($J)
    (AND (LET (QUOTE $N)
      (DECR $N))
      (SL 0 $N)))))))

(APPEND (LAMBDA ($X $Y)
  (IF (NULL $X)
    $Y
    ((LAMBDA ($W)
      ((LAMBDA ($V)
        (PROG2 (MAP $X (QUOTE (LAMBDA ($J)
          (IF (NULL (CDR $J))
            (SDR $Y $V)
            (LET (QUOTE $V)
              (RDR (LIST (CADR $J)
                $V))))))
          $W)))
        $V)))
      (LIST (CAR $X))))))

(PRIN1 (LAMBDA ($X)
  (WBCD (COND
    ((ATOM $X)
      (PNAME $X))
```

```
((NUM $X)
 (UNDEC $X))
((TRUVAL $X)
 (PNAME (IF $X (QUOTE *T)
 (QUOTE *F))))
((WBCD (PNAME (LPAREN)))
 (PNAME (RPAREN))))))

(SPACES (LAMBDA ($N)
 (AND (ENTCH TERMCH (INCR $N)
 SPBUF)
 (WBCD SPBUF)
 (ENTCH (BLANK)
 (INCR $N)
 SPBUF))))

(UNPACK (LAMBDA ($A)
 ((LAMBDA ($L $R)
 ((LAMBDA ($N)
 (UNPK1))
 (PUTBCD $R $R 1)))
 ()
 (NNAME $A))))

(UNPK1 (LAMBDA ()
 (COND
 ((EQ $N 1)
 $L)
 ((AND (LET (QUOTE $N)
 (DECR $N))
 (LET (QUOTE $L)
 (CONS (EXTCH $N $R)
 $L))))
 (UNPK1))))))

(QUOT (LAMBDA ($X)
 (LIST (QUOTE QUOTE)
 $X)))

(PRINX (LAMBDA ($X)
 ((LAMBDA ($Y)
 (TRUE))
 (PRINT $X))))
```

```
(ATOMO (LAMBDA ($A)
  (OR (NULL $A)
    (NOT (LIS $A)))))

(FACT (LAMBDA ($R)
  (PROG ($N)
    (SETQ $N 1)
    (COND
      ((ZERO $R)
        (RETURN $N)))
      (SETQ $N (MULT $R $N))
      (SETQ $R (DECR $R))
      (GO.))))

(INSIDE (LAMBDA ($A $E)
  (IF (ATOMO $E)
    (EQ $A $E)
    (OR (INSIDE $A (CAR $E))
      (INSIDE $A (CDR $E))))))

(READ (LAMBDA ($L)
  (PROG ($C)
    (COND
      ((OR (EQ (SETQ $C (NEXT*))
        (BLANK))
        (EQ $C (LPAREN))
        (EQ $C (RPAREN))))
      (RETURN (IF (NUM (DEC (PNAME (CAR $L))))
        (PROG ($N)
          (SETQ $N 1)
          (PROG ($K)
            (SETQ $K 1)
            (ENTCH TERMCH $K NUB)
            (COND
              ((EQ 9 (SETQ $K (INCR $K)))
                (RETURN (TRUE))))
              (GO.)))
          (MAPC $L (QUOTE (LAMBDA ($J)
            (PROG2 (ENTCH $J $N NUB)
              (SETQ $N (INCR $N))))))
          (RETURN (DEC NUB)))
        (PACK $L))))))
    (NCONC $L (LIST (NEXT)))
    (GO .))))
```

```
(GCD (LAMBDA ($X $Y)
  (PROG ($Z)
    (COND
      ((SL $X $Y)
        (GO .)))
      (SETQ $Z $X)
      (SETQ $X $Y)
      (SETQ $Y $Z)
    )
    (COND
      ((ZERO (SETQ $Z (REM $Y $X)))
        (RETURN $X)))
      (SETQ $Y $X)
      (SETQ $X $Z)
      (GO .))))))

(EXPT (LAMBDA ($X $N)
  (PROG ($R)
    (SETQ $R 1)
    (COND
      ((ZERO $N)
        (RETURN $R)))
      (SETQ $R (MULT $X $R))
      (SETQ $N (DECR $N))
      (GO .))))))

(EVENP (LAMBDA ($R)
  (OR (ZERO $R)
    (AND (POSV $R)
      (ZERO (REM $R 2))))))

(ODDP (LAMBDA ($R)
  (AND (POSV $R)
    (NOT (EVENP $R)))))

(DUMP (LAMBDA ($L)
  (AND (PAUSE)
    (PRIN1 (LPAREN))
    (PRIN1 (QUOTE DEFINE))
    (PRINX (LPAREN))
    (NULL (MAPC $L (QUOTE (LAMBDA ($F)
      (IF (ATOM $F)
        (PRINX (LIST $F (VAL $F)))
        (TRUE)))))))
```

```
(PRINX (RPAREN))
(PRINX (RPAREN))
(PRINX (QUOTE (PAUSE)))
(PAUSE)))

(EPRINT (LAMBDA ($X $K)
  (IF (OR (NOT (LIS $X))
    (NULL $X))
    (PRIN1 $X)
    (AND (WBCD (PNAME (LPAREN)))
      (EPRINT (CAR $X)
        (ADD $K 2))
      ((LAMBDA ($F)
        (LIS (MAP (CDR $X)
          (QUOTE (LAMBDA ($L)
            (AND (IF $F (AND (WRITE)
              (SPACES $K))
              (SPACE))
            (EPRINT (CAR $L)
              (ADD $K 2))
            (OR $F (LET (QUOTE $F)
              (LIS (CAR $L))))))))))
        (OR (LIS (CAR $X))
          (EQ (CAR $X)
            (QUOTE COND))))
        (WBCD (PNAME (RPAREN)))))))

(PRETTYPRINT (LAMBDA ($N)
  (COND
    ((LIS $N)
      (LIS (MAPC $N PRETTYPRINT)))
    ((ATOM $N)
      (AND (WRITE)
        (EPRINT (LIST $N (VAL $N))
          2)
        (WRITE)))
    ((TRUE)
      (TRUE))))))

(FIX (LAMBDA ($X $N $A $B)
  ((LAMBDA ($S)
    (LET $X (FIXA (VAL $X)))
    (FALSE))))

(FIXA (LAMBDA ($E)
  (COND
```

```
((MATCHES $E $A)
  (PROG2 (LET (QUOTE $S)
    (AND (LET (QUOTE $N)
      (DECR $N))
      (SL $N 1)))
    (MAKENEW $B (MATVARS $E $A ())))))
((LIS $E)
  (MAPCAR $E (QUOTE (LAMBDA ($J)
    (IF $S $J (FIXA $J))))))
((TRUE)
  $E)))

(MATCHES (LAMBDA ($X $P)
  (OR (EQ $P $X)
    (AND (EQ $P (ZERZER))
      (NOT (NULL $X)))
    (AND (LIS $X)
      (LIS $P)
      (MATCHES (CAR $X)
        (CAR $P))
      (MATCHES (CDR $X)
        (CDR $P))))))

(MATVARS (LAMBDA ($X $P $L)
  (COND
    ((NOT (LIS $P))
      (IF (EQ $P (ZERZER))
        (CONS $X $L)
        $L))
    ((NULL $P)
      $L)
    ((TRUE)
      (MATVARS (CAR $X)
        (CAR $P)
        (MATVARS (CDR $X)
          (CDR $P)
          $L))))))

(MAKENEW (LAMBDA ($P $V)
  (MAKEN $P)))

(MAKEN (LAMBDA ($P)
  (IF (LIS $P)
    (IF (EQ (CAR $P)
      (ZERZER))
      (CAR (NTH $V (CADR $P)))
      (MAPCAR $P MAKEN))
    $P)))
```

```

(TRACE (LAMBDA ($F)
  ((LAMBDA ($V)
    (IF (LIS $V)
      ((LAMBDA ($W)
        (IF (LIS $W)
          (COND
            ((EQ (CAR $W)
              (QUOTE TRACE1))
            (CADR (CADR $W)))
            ((AND (LIS (CADDR $W))
              (EQ (CAR (CADDR $W))
                (QUOTE CTRACE1)))
            (CADR (CADR (CADDR $W))))))
        (FALSE)))
      (CADDR $V))
    (FALSE)))
  (VAL $F))))))

(UNTRACE (LAMBDA ($L)
  (MAPC $L (QUOTE (LAMBDA ($F)
    (IF (LIS (TRACP $F))
      (LET $F (LIST (CAR (VAL $F))
        (CADR (VAL $F))
        (TRACP $F)))
      (PRINT (CONS $F (QUOTE (NOT TRACED))))))))))

(TRACE (LAMBDA ($L)
  (MAPC $L (QUOTE (LAMBDA ($F)
    (COND
      ((LIS (TRACP $F))
        (PRINT (CONS $F (QUOTE (ALREADY TRACED)))))
      ((NOT (AND (LIS (VAL $F))
        (EQ (CAR (VAL $F))
          (QUOTE LAMBDA))))
        (PRINT (CONS $F (QUOTE (NOT TRACEABLE)))))
      ((TRUE)
        ((LAMBDA ($V)
          (LET $F (LIST (QUOTE LAMBDA)
            (CADR $V)
            (LIST (QUOTE TRACE1)
              (QUOT (CADDR $V))
              (QUOT $F)
              (CONS (QUOTE LIST)
                (CADR $V))))))
          (VAL $F))))))))))

```

```

(TRACE1 (LAMBDA ($ET $FT $VT)
  (PROG2 (PRINT (CONS $FT (CONS (ZERZER)
    $VT))))
  ((LAMBDA ($QT)
    ((LAMBDA ($RT)
      $QT)
      (PRINT (LIST $FT (QUOTE ≡)
        $QT))))
    (EVAL $ET)))))

(CTRACE (LAMBDA ($FN $CD $WB $WA)
  (COND
    ((LIS (TRACP $FN))
      (PRINT (CONS $FN (QUOTE (ALREADY TRACED)))))
    ((NOT (AND (LIS (VAL $FN))
      (EQ (CAR (VAL $FN))
        (QUOTE LAMBDA)))))
      (PRINT (CONS $FN (QUOTE (NOT TRACEABLE)))))
    ((TRUE)
      ((LAMBDA ($DF)
        (LET $FN (LIST (QUOTE LAMBDA)
          (CADR $DF)
          (LIST (QUOTE $IF)
            $CD
            (LIST (QUOTE CTRACE1)
              (QUOT (CADDR $DF))
              (QUOT $FN)
              (QUOT $WB)
              (QUOT $WA))
            (CADDR $DF)))))
          (VAL $FN))))))

(CTRACE1 (LAMBDA ($EC $FC $BC $AC)
  ((LAMBDA ($WC $QC)
    ((LAMBDA ($PC)
      $QC)
      (AND (PRINX (LIST $FC (QUOTE >)))
        (LIS (MAPC $AC (QUOTE (LAMBDA ($JC)
          (PRINT (LIST $JC (QUOTE /)
            (EVAL $JC)))))))
        (PRINX (LIST $FC (QUOTE ≡)
          $QC)))))
    (AND (PRINX (LIST $FC (QUOTE <)))
      (LIS (MAPC $BC (QUOTE (LAMBDA ($JC)
        (PRINT (LIST $JC (QUOTE /)
          (EVAL $JC)))))))
        (EVAL $EC)))))

```

```
(MAPLIST (A B C)CDR)
((B C)(C)())
(MAPLIST (A B C)CAR)
(A B C)
(PACK (A B C))
ABC
(EVAL (NUM (PACK (QUOTE (1 2.3))))))
FALSE
(ASSOC D ((A B)(B C)(C D)(D E)(E F)))
(D E)
(MAPC (1 2 A 3 5 B)NUM)
(A 3 5 B)
(MAPC (A B C)ATOM)
()
(LENGTH (A (B C)(D (E))))
3
(LAST (A (B)(C (D))(E F)))
((E F))
(EQUAL (A (B)C)(A (B)C))
TRUE
(EQ (A (B)C)(A (B)C))
FALSE
(MEMB C (A B C D))
TRUE
(MEMB C (A (B C)D))
FALSE
(DREVERSE (A (B (C))(D E)F))
(F (D E)(B (C))A)
(SUBST * A (1 A B A 2))
(1 * B * 2)
(MAPCAR (1 2 3)INCR)
(2 3 4)
(MAPCAR (A B C)LIST)
((A)(B)(C))
(SUBPAIR (1 2 3 4 5)(A B C D)(K (A E I)B (H (E)B)C))
```

(K (1 E I)2 (H (E)2)3)
(CHARAC 47)
7
(CHCON P)
39
(MAPRED (2 5 3 7)ADD 1)
18
(MAPRED (A B C D)CONS (1 2))
(A B C D 1 2)
(REVERSE (A (B (C)D)E (F)))
((F)E (B (C)D)A)
(NTH (A B C D E F G)5)
(E F G)
(APPEND (A B C)(D E F))
(A B C D E F)
(UNPACK ABC)
(A B C)
(ATOMO ())
TRUE
(ATOMO 124)
TRUE
(PRINT A)
A
A
(PRINX A)
A
TRUE
(FACT 4)
24
(INSIDE A ((B (C A)(D F))))
TRUE
(GCD 2 5)
1
(GCD 4 16)
4

(EXPT 3 2)

9

(LET TRY (C (B (A D E)F A)(H (A P Q)(A K L)))

TRUE

(FIX TRY 2 (A : :)((: 2)X (: 1)))

TRUE

(VAL TRY)

(C (B (E X D)F A)(H (Q X P))(A K L))

(LET FF (LAMBDA (X)(IF (NULL X)X (APPEND (FP (CDR X))(LIST (CAR X)))))

TRUE

(CTRACE FF (SL (LENGTH X)3)(X)((QUOTE OK)))

TRUE

(FF (A B C))

(FF <)

(X / (B C))

(FF <)

(X / (C))

(FF <)

(X / ())

(FF >)

((QUOTE OK)/ OK)

(FF = ())

(FF >)

((QUOTE OK)/ OK)

(FF = (C))

(FF >)

((QUOTE OK)/ OK)

(FF = (C B))

(C B A)

(UNTRACE (FF))

()

(TRACE (FF))

()

(FF (A B C D))

(FF : (A B C D))

(FF : (B C D))
(FF : (C D))
(FF : (D))
(FF : ())
(FF ≡ ())
(FF ≡ (D))
(FF ≡ (D C))
(FF ≡ (D C B))
(FF ≡ (D C B A))
(D C B A)

C. - SEGNALAZIONI DI ERRORE

Il ROMALISP presenta tre tipi di segnalazioni di errore: il primo, destinato soltanto all'operatore, viene fornito solo mediante telescrivente, il secondo, destinato all'operatore e al programmatore viene fornito mediante telescrivente e mediante stampante, il terzo, destinato solo al programmatore viene fornito solo mediante stampante.

1. Il primo tipo comprende le segnalazioni relative a un funzionamento irregolare delle unità periferiche: la descrizione delle segnalazioni e dei provvedimenti da prendere viene fornita nell'app. D che tratta il sistema operativo.
2. Il secondo tipo comprende le segnalazioni di errori dipendenti dal programma ma che possono richiedere un particolare intervento dell'operatore (intervento che il programmatore deve fissare di volta in volta):
 - a. PAUSE: richiesta, effettuata da programma, di una sospensione temporanea del lavoro dell'interprete.
 - b. STOP: richiesta, da programma, di una sospensione definitiva del lavoro dell'interprete.
 - c. MAIN STORAGE EXAUST: esaurimento della memoria principale.
 - d. PUSH DOWN LIST EXAUST: esaurimento dello stack dell'interprete.
 - e. ILL.OUT.: richiesta di uscita di un oggetto che non è un atomo, né un numero, né una lista, né un valore di verità.
3. Il terzo tipo comprende le segnalazioni di errori di programma:
 - (ERROR 1 obj): si presenta se obj è il CAR di una forma funzionale e non è un atomo, né una lista;
 - (ERROR 2 obj): si presenta se obj è un atomo che ha se stesso come valore;
 - (ERROR 3 obj): si presenta se obj è il primo argomento di SETQ e non è un atomo;
 - (ERROR 4 obj): si presenta se obj è una etichetta di salto che non occorre nel programma;
 - (ERROR 5 obj): si presenta se obj, argomento di CAR o di CDR, non è una lista.

D. - OPERATE

1) - Inizializzazione

- a) Il programma OPERATE si trova sul nastro n.70 (I° blocco; occupa gli indirizzi da 200 a 4400).
- b) Viene caricato con OIF - NM - SG - VIA.
- c) Se non è possibile caricare con OIF da unità magnetica si carica OPERATE con OIF da FL.
- d) Viene inizializzato con SIF - SG - VIA.
- e) Si dichiara pronto con il messaggio READY OP.

2) - Impiego del programma OPERATE

Esegue le operazioni indicate nella tabella; qualunque sigla che differisca nel 1°, 3° o 4° carattere da quelle indicate determina la reinizializzazione di OPERATE.

OPERAZIONE	SIGLA	COMMENTO	REINIZ. OPERATE
RIAVVOLGIMENTO	DnMT	Riavvolge il nastro posto sull'unità n.	SI
CANCELLAZIONE TOTALE	ZnMT	Cancella il nastro posto sull'unità n.	SI
INDIETRO	VnMT	Riporta la testina dell'unità n un blocco indietro.	SI
RICERCA MARCA	SnMT	Ricerca la marca sul nastro posto sull'unità n ed arresta la testina dopo la marca.	SI
RICERCA AVANTI B.U.	FnMT	Ricerca in avanti un blocco unitario del nastro posto sull'unità n; stampa su TPG: BLOCK "XY"; arresta la testina dopo il blocco letto.	SI

(segue)

OPERAZIONE	SIGLA	COMMENTO	REINIZ. OPERATE
RICERCA INDIE- TRO B.U.	BnMT	Ricerca indietro un blocco unitario del nastro posto sull'unita' n; stampa su TPG: BLOCK "XY"; arresta la testina prima del blocco letto.	SI
LETTURA	RnMT	Legge un blocco del nastro posto sull'unita' n; se il blocco e' unitario stampa su TPG: BLOCK "XY"; altrimenti legge e carica in memoria a partire dall'indirizzo 4400; arresta la testina nell'interblocco precedente il blocco letto.	SI
REGISTRAZIONE	WnMT	Registra una copia della memoria a partire dall'indirizzo 4400 sul nastro posto sull'unita' n, preceduta da un blocco unitario in funzione di etichetta e seguita da una marca: 1) se il nastro e' all'inizio effettua le cancellazioni necessarie e dispone TPG in attesa del blocco unitario da registrare; l'operatore risponde "XY", quindi viene effettuata la registrazione della copia della memoria e quindi della marca ("spazio", "spazio"); il nastro si posiziona nell'interblocco precedente il blocco registrato. 2) Se non si e' all'inizio del nastro viene effettuata la ricerca indietro di B.U.; se non ci sono le B.U., arrivato all'inizio del nastro tutto procede come nel caso (1), se c'e' viene letto e stampato su TPG BLOCK "WZ", poi il TPG si dispone in attesa del blocco u-	SI

INIZ.
ERATE

3) - Segnalazioni fornite dal sistema.

Le loro cause e i relativi provvedimenti sono indicati in tabella:

SEGNALAZIONE	CAUSA	PROVVEDIMENTO
F. SER. MT	Unità nastro fuori servizio.	S0 - VIA
F. SUP. MT	Fine supporto unità nastro.	Regolarsi secondo i casi.
ERROR MT	Errore di lettura o registrazione dell'unità nastro.	Ripetere le operazioni (eventualmente) su altre unità.
PAUSE	Sospensione da programma.	Uno dei seguenti: a) VIA (proseguire) b) SCRIVI (registrare copia memoria e proseguire con RLIS). c) LEGGI (leggere copia memoria e proseguire con RLIS).
STOP	Fine lavoro da programma.	
PUSH DOWN LIST EXAUST	Esaurimento dello stack dell'interprete.	Seguire le direttive fornite dal programmatore caso per caso.
MAIN STORAGE EXAUST	Esaurimento della memoria.	" "
ERROR GC	Errore di garbage collector.	" "
ILL. OUT.	Uscita di oggetti non riconosciuti come oggetti LISP.	" "

4) - Impiego delle chiavi di consolle.

CHIAVE	POSIZIONE	EFFETTO
A	Sollevata	Entrata da FL.
	Abbassata	Entrata da TPG.
E	Sollevata	Uscita da TPG.
	Abbassata	Uscita su MZ.
U	Sollevata	-
	Abbassata	Stampa su TPG la statistica del garbage collector relativa all'ultima elaborazione LISP eseguite nella forma: NUMBGC"S XXXXXXXXX FR.WORDS YYYYYYYYY dove le Y rappresentano il numero di parole di memoria libere (in decimale) e le X rappresentano il numero di garbage collections effettuate durante l'ultima elaborazione.

5) - Programma UCIT per la lettura e scrittura su TPG di piccole zone di memoria.

- 1) Premere il tasto VIA.
- 2) Alla richiesta da TPG: DES. LUNGA OCT. l'operatore risponde con una descrizione lunga di cella pari e di lunghezza ottale multipla di 20.
- 3) UCIT dà il contenuto ottale della locazione.
- 4) Se il contenuto non deve essere modificato si inserisce CI e con VIA si ritorna a 1.
- 5) Se la locazione va modificata con VIA, TPG risponde CORRIGE e l'operatore batte il nuovo contenuto. Torna automaticamente a 1.

6) - Programma DUMP per la stampa su MZ di larghe zone di memoria.

1) Caricare da FL: OIF - FL - SG - VIA
SIF - SG - VIA

2) Inizializzare: EP - VIA.

3) DUMP stampa su TPG: FORNIRE LOCAZIONE PER DUMP OTTALE SU MZ.

4) L'operatore risponde con la parola doppia OOOIIIIIOOLLLLLL dove I sono le cifre di un indirizzo pari e L la lunghezza (alfanumerica) pari della locazione di stampare.

(Per stampare tutta la memoria LISP: 0000400000274000).

5) Al termine della stampa il programma torna automaticamente a 3.

I N D I C E A N A L I T I C O

ABS	Pag.	23	LETPNAME	Pag.	22
ADD	"	23	LIS	"	17
AND	"	28	LIST	"	28
APPLY	"	14-24	LPAREN	"	24
ARR	"	17	MAKEARR	"	23
ARREQ	"	17	MAKEAT	"	22
ATOM	"	17	MAP	"	24
BLANK	"	24	MATOM	"	22
BUCKET	"	22	MULT	"	23
CAR	"	18	NCONC	"	20
CDR	"	18	NEG	"	23
CxxR	"	18	NEGV	"	18
CxxxR	"	18	NEXT	"	25
CMAG	"	26	NEXT*	"	25
COMMA	"	24	NOT	"	17
COMMENT	"	30	NULL	"	17
COND	"	29	NUM	"	17
CONS	"	19	OPERATE	"	28-59
CSUBR	"	17	OR	"	28
DEC	"	26	PAUSE	"	27
DECR	"	23	PNAME	"	22
DEFINE	"	22	POSV	"	18
DIV	"	23	PRIM	"	17
EMAG	"	27	PRINT	"	26
ENT	"	23	PROCESS	"	12
ENTCH	"	23	PROG	"	33
EQ	"	17	PUTBCD	"	23
ERROR	"	27	QAR	"	19
EVAL	"	14-15-24	QDR	"	19
EXT	"	23	QUOTE	"	28
EXTCH	"	23	RAR	"	19
FALSE	"	17	RDR	"	20
GC	"	27	READIN	"	13-26
GO	"	33	REM	"	23
IF	"	28	RESTART	"	27
INCR	"	23	RETURN	"	33
LE	"	18	RMAG	"	26
LET	"	22	RPAREN	"	24

SAR	Pag.	19
SDR	"	20
SETQ	"	28
SL	"	17
SPACE	"	25
STOP	"	27
SUB	"	23
TRUE	"	17
TRUVAL	"	17
UMAG	"	27
UNDEC	"	26
VAL	"	22
WBCD	"	25
WMAG	"	26
WRITE	"	25
XAR	"	19
XDR	"	20
ZERO	"	18
ZERZER	"	25