LUCRAREA 1

Obiecte Lisp. Evaluarea formelor.Functii Lisp primitive

1. SCOPUL LUCRÃRII

Lucrarea are ca scop prezentarea principalelor elemente ale limbajului Lisp. De asemenea este prezentat modul în care este evaluatã o expresie Lisp precum si un numãr de functii predefinite de bazã.

2. CONSIDERATII TEORETICE

2.1. ATOMI SI LISTE. TIPURI DE OBIECTE.

Cele doua elemente de bazã ale limbajului Lisp sunt atomii si listele. Atomii reprezintã obiecte indivizibile si se împart în mai multe clase disjuncte:

- **atomi SIMBOLICI** - exemple**: ALFA, A, b, b19, +, p3*5**
- atomi **NUMERICI** - exemple: **7, 3.14, -98**
- atomi **SIRURI DE CARACTERE** - exemple: **"alfa","un atom"**

Atomii simbolici joaca rolul variabilelor din limbajele de programare clasice. Aceasta înseamnã cã ei pot fi legati în mod dinamic la o valoare oarecare. Ei pot avea ca valoare orice alt obiect Lisp (de exemplu un numãr, un alt atom simbolic, o lista, etc.). În orice sistem Lisp existã o serie de atomi simbolici predefiniti, unii dintre acestia desemnând functii primitive (exemple: SET, CONS etc.) altii desemnând variabile predefinite (exemple: *PRINT-LENGTH*, *PRINT-LEVEL* etc.) puse la îndemânã de sistemul Lisp respectiv.

Listele sunt delimitate de paranteze si pot avea ca elemente orice alt tip de obiect Lisp. Aceasta înseamnã cã o listã poate avea ca elemente atomi sau alte liste, încuibãrirea fiind permisã pe oricâte nivele.

Exemple:

(O "L I S T A" cu 5. atomi)
(UN_atom (2 1) ((alta lista)))
()

Lista vidã se simbolizeazã prin '()' sau 'NIL'. Este singurul element al limbajului Lisp care este si listã si atom în acelasi timp! Listele nu au dimensiune fixã, ele pot creste sau descreste în mod dinamic.

În Common Lisp tipurile obiectelor sunt aranjate într-o ierarhie. De exemplu

- Tipul **LIST** are ca subtip tipul **CONS**.
- Tipul **ATOM** are ca subtipuri tipurile **NUMBER, SYMBOL** si **STRING**.
- Tipul **NUMBER** are ca subtipuri tipurile **INTEGER** si **FLOAT**.
- Tipul **INTEGER** are ca subtip tipul **CHARACTER**.
- În afarã de acestea mai existã tipuri ca **ARRAY, STRUCTURE, HASH_TABLE**, etc.

2.2. EVALUAREA FORMELOR

Interpretorul Lisp executã la infinit o buclã în care citeste o expresie Lisp, încearcã sã o evalueze si, în caz de succes, afiseazã rezultatul evaluãrii, altfel un mesaj de eroare. Prin "formã" vom întelege orice expresie Lisp "bine formatã", în sensul cã ea poate fi evaluatã cu succes de cãtre interpretor.

Orice atom este o formã; o listã trebuie însã sã respecte anumite restrictii pentru a fi "formã". Atomii se evaluează dupã cum urmează:

- atomii numerici - la numãrul pe care îl reprezintã
- atomii sir de caractere - la sirul de caractere
- atomii simbolici - la valoarea la care sunt legati în acel moment.

Încercarea de evalua un atom simbolic care nu a fost legat la nici o valoare produce eroare (legarea unui simbol la o anumitã valoare se face cu functia de sistem SETQ).

În exemplele de mai jos " * " reprezintã prompterul interpretorului.

EXECUTIE	REZULTAT
*7	7
*T	T
*"ha ha ha"	"ha ha ha"
*NIL	NIL
***alfa**	ERROR: Unbound variable: ALFA
*(SETQ alfa 10)	10
***alfa**	10

Dupã cum se observã din exemplele de mai sus, atomii NIL si T sunt predefiniti si sunt legati la ei însisi. Orice încercare de schimbare a valorii lor este semnalatã ca eroare. Pentru ca o listã sã fie o formã trebuie ca primul element sã desemneze o functie, iar celelalte elemente sã fie argumentele asteptate de functia respectivã.

În Lisp întotdeauna **numele functiei stã în fata argumentelor**, iar **între functie si argumentele sale nu se utilizează paranteze!**

Expresiile de mai jos sunt corecte:

EXECUTIE	REZULTAT
***(+ 2 3 4)**	**9**
***(MAX 2 1 8 53 17)**	**53**
***(LENGTH '(a b c d))**	**4**

Exemple de expresii ce NU SUNT FORME: (2 + 3), (MAX (2 3 1))

În general procesul de **EVALUARE A UNEI FUNCTII** are loc în douã etape:

1. **întâi se evaluează argumentele functiei**
2. **asupra argumentelor evaluate se aplicã functia respectivã.**

Majoritatea functiilor se comportã în acest mod (exemple: SET, CONS, CAR, LIST, APPEND,+ etc.). Existã însã si exceptii, una dintre acestea fiind chiar functia SETQ, care nu-si evaluează decât al doilea argument. Caracterul **apostrof "'"** aflat **în fata unei expresii opreste evaluarea acelei expresii**.

Exemple:

EXECUTIE	REZULTAT
*(SETQ alfa 345)	**345**
***'alfa**	**alfa**
***alfa**	**345**
*(+ alfa 5)	**350**
*'(2 + 3)	**(2+3)**
*(2 + 3)	ERROR: Bad function: 2

Întrucât înainte de aplicarea functiei se evaluează argumentele, care pot fi alte forme ce implică alte aplicări de functii, procesul de evaluare este recursiv, oferind posibilitatea compunerii functiilor.

Exemplu:

EXECUTIE	REZULTAT
*(+ 2 (- 5 1) 3)	**9**
*(SETQ a (+ 1 (SETQ b 2)))	**3**
*(+ a b)	**5**

2.3. FUNCTII LISP PRIMITIVE

În afară de valoare si independent de aceasta, oricărui simbol poate să i se asocieze o functie definită de utilizator. Standardul Common Lisp pune însă la dispozitie un număr impresionant de mare de functii predefinite ce pot fi utilizate direct sau în definirea de noi functii. În Lisp toate functiile întorc o valoare. Pentru a fi apelate, simbolul ce desemnează functia trebuie să se afle pe prima pozitie a unei liste (forme). Numele de functii sunt simpli atomi simbolici - de aceea, evaluarea unui simbol ce reprezintă numele unei functii nu implică invocarea functiei, ci simpla întoarcere a eventualei valori asociate. Prezentăm în continuare câteva dintre functiile mai importante:

2.3.1. FUNCTII ARITMETICE SI LOGICE

+, -, *, /, MIN, MAX, TRUNCATE, LOGAND, LOGIOR, LOGNOT, LOGXOR desemnează functiile aritmetice si logice (pe biti) uzuale. Cu exceptia lui TRUNCATE si LOGNOT, ele pot avea oricâte argumente. Exemple:

EXECUTIE	REZULTAT
***(* 1 2 3 4)**	**24**
***(- 10 1 2 3)**	**4**
***(LOGIOR 2 4 8)**	**14**

2.3.2. SELECTORI

Limbajul Lisp oferă puternice facilităti de prelucrare a listelor. Principalele **primitive** care selectează elemente dintr-o listă sunt:

FIRST (sau CAR)	-primul element al listei;
REST (sau CDR)	-lista obtinutã dacã se eliminã primul element;
SECOND (sau CADR), THIRD (sau CADDR)	- al doilea, al treilea element;
simbolurile cu structura: "C" urmat de un numãr oarecare de "A" si/sau "D" si terminate cu "R"	-prescurtare a compunerilor de CAR si CDR, -Exemplu: (CDADR <expr>) <=> (CDR (CAR (CDR <expr>))). -Nu se pot compune în acest mod mai mult de 3 functii!
LAST	-întoarce lista ce contine doar ultimul element al unei liste;
NTH	-întoarce elementul al n-lea (primul element are indicele 0);
NTHCDR	-întoarce sublista ce începe la indicele n.

Exemple:

EXECUTIE	REZULTAT
*(FIRST '(a b c))	A
*(REST '(a b c))	(B,C)
*(FIRST (REST '(a b)))	B
*(CADR '(a b c))	B
*(NTH 2 '(a b c))	C
*(LAST '(a b c d))	(D)

2.3.3. CONSTRUCTORI

Listele reprezintã structuri de date dinamice. Functiile din aceastã categorie construiesc noi liste sau adaugã noi elemente la liste existente. Aceste operatii sunt principalele consumatoare de memorie în Lisp.

Observatie: Aceste operatii presupun în limbajele de programare procedurale clasice folosirea unor functii ca "new" în Pascal, sau "malloc" în C. Alocarea si eliberarea zonelor de memorie este o operatiune transparentã în Lisp, sistemul având grijã sã elibereze zonele ce nu mai sunt referite printr-un mecanism numit "**garbage collector**".

Principalii **CONSTRUCTORI** sunt:

CONS	- asteaptã 2 argumente. Dacã al doilea argument se evalueazã la o listã, atunci adaugã primul argument ca prim element al listei si întoarce noua listã.
APPEND	- asteaptã un numãr nedefinit de liste, întoarce o listã care rezultã prin concatenarea lor (se creeazã copii ale primelor n-1 argumente).
LIST	- asteaptã un numãr nedefinit de obiecte Lisp, construieste o listã ce are ca elemente argumentele primite.
REVERSE	- întoarce o listã cu elementele listei argument luate în ordine inversã.
REMOVE	- asteaptã douã argumente, dintre care al doilea trebuie sã fie listã. Întoarce o copie a listei din care sunt eliminate aparitiile la nivel superficial ale primului argument.
SUBST	- asteaptã 3 argumente, al 3-lea trebuind sã fie listã. Întoarce o copie a listei în care toate aparitiile celui de-al doilea argument (de pe orice nivel de imbricare) sunt înlocuite cu primul argument.

Exemple:

EXECUTIE	REZULTAT
*(SETQ a '(1 2))	**(1 2)**
*(CONS 0 a)	(0 1 2)
*a	(1 2)
*(APPEND a '(3 4) a)	(1 2 3 4 1 2)
*(LIST a '(3 4) a)	((1 2) (3 4) (1 2))
*(REMOVE 1 '(1 (1) a))	((1) A)
*(REVERSE '(1 2 3))	(3 2 1)
*(LIST 'a 'b 'c)	(A B C)
*(CONS '(a) '(b c))	((A) B C)

2.3.4. LEGARE

Principalele primitive prin intermediul cãrora se poate lega o valoare la un simbol sunt SETQ, SETF si SET. Ele asteaptã douã argumente, cel de-al doilea fiind valoarea la care se face legarea. Al doilea parametru se evaluează întotdeauna, valoarea rezultatã fiind si cea întoarsã de formele SET. Cele 3 functii diferã între ele prin modul în care tratează primul parametru:

SETQ	- primul parametru nu se evaluează si el trebuie sã fie un atom simbolic, altul decât T sau NIL. Valoarea se leagã la respectivul simbol.
SET	- (legare indirectã) primul parametru se evaluează, în urma evaluării trebuind sã se întoarcã un atom simbolic, la care se va face atribuirea
SETF	- (legare prin referintã) este asemãnãtoare cu SETQ, în sensul cã primul parametru nu se evaluează, dar este mult mai generalã. SETF este o functie generică. Ea "stie" sã schimbe valoarea oricãrui câmp ce poate fi referit. Primul argument trebuie sã reprezinte o formã care, daca s-ar evalua, ar întoarce valoarea unui anumit câmp. Respectivul câmp este cel setat de SETF.

Exemple:

EXECUTIE	REZULTAT
*(SETF alfa 'a)	**a**
*alfa	**a**
*a	ERR: Unbound var: a
*(SET alfa 'beta)	**beta**
***alfa**	**a**
***a**	**beta**
*(SET (CAR '(a b)) 5)	5
*a	5
*(SET (CAR '(1 2)) 5)	ERR: SYMBOL expected

2.3.5. PREDICATE

Simbolurilor T si NIL li se asociazã în Lisp semnificatia de TRUE, respectiv FALSE. Unele predicate în loc de T întorc o expresie oarecare diferitã de NIL (exemple: MEMBER, OR, AND). Existã diverse clase de predicate: ce testeazã relatii între numere, ce testeazã relatii între siruri de caractere, ce testeazã identitatea sau izomorfismul structural al obiectelor, ce testeazã apartenenta unei expresii la o listã, ce testeazã apartenenta unui obiect Lisp la un tip sau subtip.

Le vom prezenta în aceastã ordine.

ZEROP	- are argumentul valoarea zero ?
PLUSP	- este argumentul un numãr pozitiv ?
MINUSP	- este argumentul un numãr negativ ?
EVENP	- este argumentul un numãr par ?
= **>** **<**	- sunt toate argumentele egale între ele ? -sunt argumentele aranjate în ordine descrescãtoare ? - sunt argumentele aranjate în ordine crescãtoare ?

EXECUTIE	REZULTAT
*(< 1 3 5 7)	T
*(= 1.0 1)	T
*(> 5 2 3)	NIL

STRING-EQUAL (sau STRING=)	- sunt cele doua siruri de caractere egale, formate din aceleasi caractere, în aceeasi ordine? (Pentru testarea identitãtii se foloseste EQ sau EQL.)
STRING-LESSP (STRING<)	- dacã primul sir de caractere este lexicografic mai mic decât al doilea, se întoarce indicele caracterului din primul sir care este mai mic decât cel corespondent din al doilea, altfel se întoarce NIL.
CHAR-LESSP	- sunt argumentele caractere în ordine alfabetica
ALPHA-CHAR-P	- este argumentul caracter alfanumeric
CHAR-EQUAL	- reprezintã toate argumentele aceeasi literã (literele mari sau mici sunt tratate identic)

EXECUTIE	REZULTAT
*(STRING= "abc" "abc")	T
*(STRING< "aaa" "aab")	2

ENDP (sau NULL)	- întoarce T daca argumentul e o listã vidã, NIL în caz contrar.
EQ	- întoarce T dacã cele douã argumente desemneazã acelasi obiect Lisp. Poate fi aplicat pe orice tip de argumente.
EQUAL	- întoarce T dacã cele douã argumente desemneazã obiecte cu aceeasi structurã. Douã argumente care sunt "egale" conform lui EQ vor fi egale si conform lui EQUAL, reciproca nefiind adevãratã.

EXECUTIE	REZULTAT
*(ENDP ())	T
*(NULL '(a b))	NIL
*(EQ 'a 'b)	NIL
*(EQ '(a b) '(a b))	NIL
*(EQUAL '(a b) '(a b))	T
*(EQUAL 1 1.0)	NIL

	- testează dacă primul argument este element în lista reprezentată de cel de-al doilea argument. În caz afirmativ se întoarce sublista ce începe cu acel element, altfel se întoarce NIL. Testarea egalitătii se face implicit cu EQ, dar se poate specifica orice predicat binar cu ajutorul cuvântului cheie :TEST. Căutarea se face doar la nivel superficial.

MEMBER

EXECUTIE	REZULTAT
*(MEMBER 'MOTHER '(TELL ME MORE ABOUT YOUR MOTHER PLEASE))	(MOTHER PLEASE)
*(MEMBER 'MOTHER '((FATHER SON)(MOTHER DAUGHTER)))	**NIL**
*(MEMBER '(A B) '((A B) (C D)))	**NIL**
*(MEMBER '(A B) '((A B) (C D)) :TEST #'EQUAL)	((A B) (C D))

TYPEP	- testează apartenenta obiectului reprezentat de primul argument la tipul reprezentat de al doilea argument.

Corespunzător cu fiecare tip de date existent în Common Lisp sistemul defineste implicit niste predicate ce testează apartenenta unui obiect la acel tip.

Cu exceptia predicatului ATOM, celelalte sunt formate prin adăugarea literei "P" la simbolul ce reprezintă numele tipului (exemple**: SYMBOLP, NUMBERP, LISTP, STRINGP, CHARACTERP, INTEGERP, FLOATP, ARRAYP, CONSP).**

EXECUTIE	REZULTAT
*(ATOM 'ALFA)	**T**
*(ATOM ())	**T**
*(ATOM '(a b))	**NIL**
*(SYMBOLP 2)	**NIL**
*(CHARACTERP #\a)	**T**
***(TYPEP 'ALFA 'SYMBOL)**	**T**

Predicatele pot fi combinate cu ajutorul functiilor AND, OR, NOT. Datorită faptului că AND si OR nu-si evaluează toate argumentele, ci doar atâtea câte sunt necesare pentru deducerea valorii de adevăr, ele mai sunt folosite si pentru controlul evaluării.

2.3.6. FUNCTII DE INTRARE/IESIRE

TERPRI	- trimite un caracter new-line la iesirea standard
PRINT, PRIN1, PRINC	- scriu în fisierul standard de iesire valoarea argumentului. Spre deosebire de celelalte două, PRINT trimite un caracter new-line înaintea scrierii. Spre deosebire de celelalte două, PRINC nu scrie si caracterele ghilimele (") la tipărirea argumentelor de tip STRING
PPRINT	- scriere indentată, "frumoasă", a argumentului
READ	- asteaptă introducerea de la consolă a unui obiect Lisp (atom, listă, formă, vector, etc.), pe care îl întoarce neevaluat.

(FORMAT <canal> <format>)	**- folosită pentru scriere cu format conform sintaxei: după al doilea argument trebuie să urmeze atâtea argumente câte sunt specificate în format. <canal> specifică locul unde se face scrierea (T specifică iesirea standard, NIL indică scrierea într-un sir de caractere). Dintre specificatorii de format - introdusi de caracterul "~" - amintim:** ~A - scrie argumentul următor care poate fi orice obiect ~D - scrie argumentul următor care trebuie să fie număr ~% - trece la linie nouă

Exemple:

EXECUTIE	REZULTAT
***(PRINT '(PLEASE PRINT THIS)) ;trece la linie nouă**	
(PLEASE PRINT THIS) ;efectul functiei PRINT	(PLEASE PRINT THIS) ;valoare întoarsă
***(TERPRI) ;efect lateral TERPRI**	NIL ;valoare intoarsa
***(SETF input (READ)) ; se introduce: (table chair (pen))** **(TABLE CHAIR (PEN))**	
***input**	(TABLE CHAIR (PEN))
***(FORMAT T "~%The list ~A has ~D elements." input (LENGTH input))**	The list (TABLE CHAIR (PEN)) has 3 elements.

3. DESFÃSURAREA LUCRÃRII

1. Se vor testa toate functiile prezentate în lucrare.

2. Sã se indice tipul elementelor de mai jos si apoi sã se verifice cu ajutorul predicatelor: ATOM, SYMBOLP, STRINGP, NUMBERP, LISTP, CHARACTERP.

a) CAR b) Un_cuvant*ceva? c) oare ce9 d) 9ec e) "a a a"
f) () g) > h) (a s (d k)) i) (NIL ())
j) (((j))) k) 97 l) #\A

3. Sã se evalueze manual formele de mai jos si apoi sã se verifice:

(ATOM NIL)	(LISTP NIL)	(SYMBOLP NIL)	(NULL NIL)
(ATOM '(a b))	(LISTP '(a b))	(NULL '(a b))	(NULL (REST '(b))
(ATOM 'alfa)	(LISTP 'alfa)	(SYMBOLP 'alfa)	
(ATOM "a a")	(SYMBOLP "a a")	(TYPEP 1.0 'INTEGER)	

4. Sã se scrie în Lisp formele ce calculeazã expresiile :

a) 2 + 3 + (20 - 5 * 6 / 2) / 5 b) (4 - 2 + 3 * 5 / 7)* 125 c) 11 - 3 - (43 + (15 - 11)*2 +1)

5. Sã se evalueze manual si apoi cu ajutorul interpretorului formele:

(SETF a (+ 3 2 5)) (+ a (* 3 2 6)) (SETF b (+ 10 a)) a (SETF a (SETF b (+ a b))) a b	(CONS 'a '(b c)) (CONS '(b c) '(a)) (CONS NIL NIL) (CONS 'a (CONS '(b) NIL)) (CONS 'a 'b) (CAR '(a b c)) (FIRST '(a b c)) (CDR '(a b c)) (REST '(a b c)) (SECOND '(a b c))	(SETF a '(((1) (2)) ((3)) 4)) a (CAR a) (CAR (CAR a)) (CAR (CDR a)) (CADR a) (CDDR a) (CADAR a) (THIRD a) (CAAR a) (NTH a 2) (LAST (CAR a))
(LENGTH a) (LENGTH (CAR a)) (CONS (CAR a) a) (LENGTH (CONS (CAR a) a)) (MEMBER a a) (MEMBER 'me '(how are you)) (MEMBER 'nice '(what a nice day)) (MEMBER 'me '((you) and (me)))		(SETQ p '(l i s 1)) (SETQ q '(l i s 2)) (LIST p q) (APPEND p q) (APPEND NIL NIL NIL) (LIST NIL NIL NIL) (APPEND t NIL p) (LIST 'alfa) (REVERSE p) (REVERSE (REVERSE p)) (REVERSE (LIST p q s))
(SETF p '(un mar si o para)) (SUBST 'niste 'o (SUBST 'struguri 'para p))		(> 1 3 5.1) (< 1 3 5.1) (> 1 5 3) (< 1 5 3) (MIN 1.1 10 2.5) (MAX 1.1 10 2.5)

6. Sã se scrie formele care selectează atomul 'alfa' din listele:

a) (o (lista (alfa))) b) (((o) lista) alfa) c) (este (ceva (mai jos : (alfa))))

7. Sã se indice care dintre expresiile de mai jos produc eroare dacã se evaluează si de ce :

a) (APPEND 'a 'b 'c) b) (APPEND NIL NIL) c) (LIST 'a 'b 'c) d) (LIST 'x)
e) (MAX (1. '2. 3.)) f) (MIN '(1. 2. 3.)) g) (> '(a b c) '(a b))
h) (< (LENGTH '(a b c)) (LENGTH '(a c))) i) (SETF 'alfa 'beta) j) (+ 2. 'a)