Programare Funcționala

Lucrarea 1

INTRODUCERE

Limbajele de programare sunt de doua tipuri:

- procedurale – imperative (indica cum se rezolvă o problemă - pas cu pas);
- neprocedurale – funcționale (indică subproblemele ce trebuie rezolvate).

Programarea funcțională este o paradigmă de programare care consideră calculul ca o evaluare a funcțiilor matematice și evită schimbarea stării și a datelor modificabile.

Programarea imperativă este o paradigmă de programare care folosește instrucțiuni, care schimbă starea unui program.

LISP este un limbaj neprocedural, funcțional, al doilea ca vechime după FORTRAN. A fost inventat de către John McCarthy în 1958. În perioada 1965 - 1975 s-au căută noi variante. La începutul anilor 1970 existau două dialecte: MacLisp şi Interlisp.

Maclisp s-a dezvoltat din Lisp 1.5 prin adăugarea variabilelor speciale şi a tratării erorilor.

Interlisp a introdus mai multe concepte noi, contrucţia iterativa, preluată de MacLisp şi mai târziu de CommonLisp.

În 1981 se impune COMMON LISP, care este implementat în 1984 ca descrierea unei familii de limbaje.

În cadrul laboratorului de **Programare Funcțională -PF-** vom folosi pachetul XLISP-stat sau CLisp se sub Linux. Salvaţi xlisp-stat pentru Windows de la ftp://ftp.stat.umn.edu/pub/xlispstat/ versiunea cea mai recentă.

LISP este limbajul care se bazează pe programarea funcțională, conceptul de prelucrare a datelor fiind cunoscut sub numele de prelucrare simbolică.

O funcție LISP este un obiect matematic care întoarce o unică valoare pentru un set de valori de intrare. Funcțiile întorc valori atunci când sunt evaluate.

Următoarele simboluri au o semnificație specială în Lisp:

(– o paranteză stângă marchează începutul unei liste;

) – o paranteză dreaptă marchează sfîrşitul unei liste;

' – un apostrof, urmat de o expresie ***e, 'e***, reprezintă o scriere condensată pentru un apel (quote e); se mai folosește pentru a marca începutul unui mesaj ce urmează a fi tipărit;

; – punct-virgula marchează începutul unui comentariu. El însuși împreună cu toate caracterele care urmează până la sfârșitul rândului sunt ignorate;

" – între o pereche de ghilimele se include un şir de caractere;

LISP este un limbaj pentru prelucrarea listelor. Procedurile şi datele au aceeaşi structură, şi anume o listă generalizată. Elementele unei liste pot fi simboluri, ce definesc **ATOMII**, și alte **LISTE**, formând aşa numită **expresie simbolică.**

Elementele unei liste sunt separate sintactic printr-un singur spațiu.

Implementările uzuale de Common Lisp sunt insensibile la forma caracterelor (minuscule sau majuscule). Intern însă, formele Lisp sunt reprezentate cu majuscule, de aceea formele rezultate în urma evaluărilor sunt redate în majuscule.

EXEMPLE DE EXPRESII SIMBOLICE:

(a alfa b (c d e) delta) – listă cu 5 elemente, 4 atomi simbolici şi o listă cu 3 elemente atomi simbolici

(3 4 5 (6 7)) – listă cu 4 elemente, 3 atomi numerici şi unul , listă cu 3 elemente atomi numerici

Tipurile de date elementare în LISP sunt:

Numeric Număr întreg: 56, 4 , -312 etc.
Număr real: 45.3, 1.5, -13.2 etc.
Număr complex - secvenţa:#c(<real> <imaginar>) :#c(1 2)

ATOM

Simbolic: A, L, Suma, Alpha, MAX, +, *, /, etc.

Orice combinație de litere și cifre poate constitui un nume de **simbol**. În plus, oricare dintre următoarele caractere poate interveni într-un nume de simbol: + - * / @ $ % ^ & _ = < > ~

LISTA – expresie de forma: (elem1 elem 2 … elem n)

Lista poate conține oricâte elemente de tip atom sau listă separate printr-un singur spaţiu

LISP-ul ignoră spațiile în plus sau ENTER

Dacă toate parantezele deschise nu sunt închise de utilizator, LISP-ul aşteaptă în continuare paranteză închisă. (ATENTIE!!! Expresia nu poate fi evaluată)

Se pot pune mai multe paranteze dreapta decât stânga, LISP-ul le va ignora pe cele în plus.

EXEMPLE:

- () – lista vidă, notată şi nil sau NIL;
- (a atom 3) – lista formată din 3 atomi, dintre care primii doi simbolici, iar al treilea numeric;
- (atom (a) elem (b 1) lista (c 1 2)) – lista formată din 6 elemente: primul, al treilea şi al cincilea fiind atomi simbolici, iar al doilea, al patrulea şi al şaselea – fiind la rândul lor liste cu unu, două, respectiv trei elemente, câte unul nenumeric, restul numerici.
- ((ATOM) Atom)))) – parantezele stânga în plus sunt ignorate, expresie simbolică corectă

În LISP operăm cu următoarele categorii de obiecte:

- **Variabile**
- **Constante**
- **Date**
- **Funcții**
- **Macrouri**

Numai datele au tipuri, o variabilă în LISP nu are definit un tip. Ea poate primi ca valoare orice tip de dată.

Există trei moduri prin care o variabilă poate primi valori:

- **prin asignare;**
- **prin legare;**
- **prin listă de proprietăți.**

Constantele sunt simboluri care au atașate valori ce nu pot fi modificate.

În Lisp nu există proceduri ci numai funcţii, în sensul că orice rutină întoarce obligatoriu şi o valoare.

Macro-urile sunt funcţii care au un mecanism de evaluare special, în doi paşi: expandarea şi evaluarea propriu-zisă.

Un program Lisp este format numai din apeluri de funcții. Practic, însăși o definiție de funcție este, de fapt, tot un apel al unei funcții care creează o asociere între un nume al funcției, o listă de parametri formali şi un corp al ei.

Expresiile Lisp sunt interpretate într-o buclă **READ – EVAL – PRINT**, în care are loc o fază de citire a expresiei de pe fluxul de intrare, urmată de evaluarea expresiei, şi o ultimă fază de tipărire a rezultatului.

Read – citeşte o expresie simbolică;

Eval – evaluează expresia simbolică introdusă;

Print – afişează rezultatul obţinut în urma evaluării expresiei.

Orice expresie ce urmează a fi evaluată se prezintă pe un rând imediat după *prompter* (>), pentru ca pe rândul următor să apară rezultatul tipărit al evaluării.

În Lisp pot fi reprezentate numere oricât de mari fără riscul de a provoca vreo depășire:

>9999999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999999999999999999999

- un atom simbolic care are o valoare predefinită (în sensul asignării sau al legării) se evaluează la această valoare.

Expresiile se evaluează după câteva reguli simple:

- un atom numeric se evaluează la el însuși:

```
> 3
3
```

Astfel, presupunând că simbolului **a** i se asociase anterior valoarea **alpha**:

```
> a
alpha
```

Excepție fac simbolurile: **nil** care, fiind notația pentru lista vidă cât şi pentru valoarea logică fals, se evaluează la el însuşi , şi **t** care se evaluează la valoare logică adevărat (true sau TRUE).

Orice expresie diferită de nil este considerată a fi echivalentă logic cu true:

```
> nil
NIL
> t
TRUE
```

Un atom simbolic care nu are o valoare predefinită, tentat a fi evaluat, va provoca un mesaj de eroare de genul UNBOUND-VARIABLE;

O expresie simbolică prefixată cu apostrof (*quote*) se evaluează la ea însăşi:

```
> ’alpha
alpha
> ’3
3
> ’(a 3 ())
(a 3 NIL)
```

O listă care are primul element o forma LISP predefinită (operatori aritmetici, funcții matematice, primitive LISP) sau o funcție definită de utilizator, evaluează restul elementelor listei conform formei respective.

EXEMPLE:

```
> (+ 2 1)
3
>(+  (* 3 2)  (- 8 2))
12
>(expt  3  2)
9
```

FUNCȚII PREDEFINITE

Funcții de intrare

Funcția de intrare standard în Lisp este funcția **READ.** Dacă funcția nu are nici un argument, citirea se face de la tastatură.

Exemplu:

```
> (read)
3456 <ENTER>
3456
> (read)
"cati ani ai" <ENTER>
"cati ani ai"
```

După apelul funcției se așteaptă citirea pe o durată nelimitată, până la introducerea de la tastatură, validarea citirii se face cu ENTER, apoi se returnează valoarea citită.
Citirea în Lisp nu presupune numai citirea caracterelor și returnarea lor ca șir de caractere, ci Lisp-ul analizează ceea ce citește și returnează obiectul Lisp rezultat.

Funcții de ieșire

Cea mai generală **funcție de ieșire** în Lisp este funcția **FORMAT**. Ea are două sau mai multe argumente:

primul indică unde va fi tipărit rezultatul;

al doilea este un șablon șir de caractere;

iar restul argumentelor sunt de obicei obiecte ale căror reprezentări tipărite sunt inserate în șablon.

Exemplu:

```
> (format t "~A plus ~A egal ~A.~%" 2 3 (+ 2 3))
2 plus 3 egal 5.
NIL
```

Funcția returnează pe prima linie ceea ce afișează FORMAT, iar pe a doua linie se afișează valoarea returnată de apelul Format. Atunci când funcția este utilizată în interiorul programelor, valoarea de pe a doua linie nu este vizibilă. Primul argument al funcției t, indică faptul ca afișarea va fi făcută în locul implicit. Al doilea argument este un șir care servește drept șablon pentru afișare. În cadrul acestui șir, fiecare caracter ~A indică o poziție care va fi ocupată, iar ~% indică o linie nouă. Pozițiile vor fi ocupate de valorile argumentelor următoare în ordinea în care ele apar în Format.

Funcții aritmetice

Lisp-ul are o biblioteca foarte bogată de funcții aritmetice. Vom prezenta numai câteva dintre acestea:

+ adunare: se pot aduna oricâte argumente:

```
> (+ 2 3 4 5)
14
```

- scădere: dacă avem un singur argument, atunci se obține numărul negativ:

```
> (- 7)
-7
> (- 8 3 2)
3
```

*** inmulțire:**

```
> (* 3 5 1)
15
```

/ impărțire:

```
> (/ 6 3)
2
> (/ 75 3 5)
5
```

Dacă argumentele funcțiilor aritmetice sunt întregi, rezultatul va fi tot întreg.
Dacă unul din argumente este real, atunci rezultatul va fi real

```
> (+ 6 (- 7.5 3))
10.5
```

Excepție este dacă împărțim două numere întregi care nu dau un rezultat exact, atunci rezultatul este un număr real.

```
> (/ 25 4 3)
2.08333
```

Toate operațiile de mai sus se aplică şi argumentelor numere complexe:

```
> (+ #c(1 2) #c(3 2))
#C(4 4)
> (- #c(1 2) #c(3 2))
-2
> (* #c(1 2) #c(3 2))
#C(-1 8)
> (/ #c(1 2) #c(3 2))
#C(0.538462 0.307692)
```

Funcțiile **floor** și **ceiling** , rotunjesc o valoare în jos, respectiv în sus.

```
> (floor 2)
2
> (floor 4.66)
4
> (floor 2.33)
2
> (ceiling 2.99)
3
> (ceiling 2.25)
3
> (floor (* 2 (/ 5 3)))
3
> (ceiling (* 2 (/ 5 3)))
4
```

Pentru numerele întregi, există două funcții care returnează restul împărțirii celor două numere: Funcțiile **REM** sau **MOD.** Ambele au același efect.

```
> (rem 5 3)
2
> (mod 14 5)
4
```

Există două forme LISP de **incrementare,** respectiv **decrementare**, fără efecte laterale:
Funcțiile **1+** respectiv **1-**

```
> (1+  5)
6
> (1-  8)
7
```

Funcția de determinare a valorii absolute: **ABS**

```
> (abs 5)
5
> (abs -3)
3
```

Dacă argumentul este număr complex, rezultatul este modulul:

```
> (abs  #c(3  -4))
5
```

Funcţia radical**: SQRT** - are un singur argument

```
> (sqrt  3)
1.73205
> (sqrt 5 4)
error: too many arguments
```

Funcţia ridicare la putere **EXPT –** are 2 argumente, primul este baza, iar al doilea este puterea

```
>(EXPT 2 3)
8
```

Funcţiile **MIN** si **MAX –** calculează minimul şi maximul unui şir de **n** numere:

```
>(MIN 5 2 -3 7 8)
-3
>(MAX 2 1 8 3 7)
8
```

PROBLEME

1. Precizați numărul de elemente ale următoarelor liste, și apoi tipul elementelor:

((1 3) 3.4 (a b c) ((atom)))
((a (b c)) d)
((((a))) ((b)) (c) d)
(a b cd)

2. Scrieți exemple de liste :

Cu 4 elemente atomi de tip numeric
Cu 3 elemente de tip listă
Cu 2 elemente atomi simbolici și 2 elemente de tip listă, fiecare cu câte 2 elemente atomi numerici

3. Transpuneți în LISP următoarele expresii şi apoi evaluați-le:
   (25 + 30) * 15/2
   6 * 3.1416

4. Determinaţi câtul întreg pentru media aritmetică a numerelor:
   5, 6.7, -23.2, 75 și 100.3

5. Determinaţi restul împărțirii numerelor:
   365 12, 13 şi 467

6. Scrieţi sub formă de expresie LISP soluțiile următoarei ecuații de gradul 2, apoi evaluați expresiile:
   $2x^2 + 7x + 5 = 0$

7. Determinaţi rezultatul întreg prin rotunjire în jos, respectiv în sus, apoi incrementați, respectiv decrementați rezultatul pentru următoarea expresie: (5 * 2.25 – 7.13) / (45 – 25 / 5)

8. Evaluaţi următoarele forme:
   (* (MAX 3 4 5) (MIN 3 4 5))
   (EXPT (MAX 3 1 4) (MAX 2 7 1))
   (REM (+ 5 7 13) (- 14 1))

9. Afişaţi rezultatul de la problema 3 sub forma unui mesaj:
   Ex: restul impărțirii numerelor … şi … este:…

10. Afișați un mesaj înainte de citirea unei variabile, apoi citiți variabila.
    Ex: “a =” citire valoare