Lab 2

L I S P

Topics

In LISP, procedures and data have the same structure: **generalized list.**

- the fundamental objects are called ATOMS;
- groups of atoms and lists form LISTS;
- atoms and lists are called SYMBOLIC EXPRESSIONS.

LISP means manipulating symbols.

In LISP, the procedure is always specified at the head of the list, immediately followed by arguments => prefix notation.

- a PROCEDURE is a basic entity that specifies the way something is done;
- a procedure predefined by the language is called a PRIMITIVE
- a collection of procedures that work together are called a PROGRAM
- an abstract description of a procedure or a program, that is not expressed in any specific coding language, is called an ALGORITHM.

When a right paranthesis and a left paranthesis surround something, we call the result a list and we reffer to its elements.

For example, the list `(+ 3.14 2.71)` has 3 elements: +, 3.14 şi 2.71.

LISP examples

LISP evaluates a list assuming that the first element is a procedure and the rest of the elements are parameters.

```
(+ 3.14 2.71)
5.85
(* 9 3)
27
(/ 27 3)
9
(MAX 2 4 3)
4
(MIN 2 4 3)
2
(EXPT 2 3) ; exponentiation
8
(SQRT 4.0) ; square root
2.0
(ABS -5)
5
(- 8)
-8
```

Considering the expression:

```
(+ (* 2 2) (/ 2 2))
```

We can easily notice that it is evaluated to 5.

- The atoms of the type: 27, 3.14 are called NUMERICAL ATOMS.
- The atoms of the type: +, MAX are called SYMBOLICAL ATOMS.

Forming an expression in LISP:

```
  Expression
      /\
     /  \
    /    \
   /      \
         /       \
      Atom      List
                  /\
                 /  \
                /    \
               /      \
        Number    Symbol
            /\
           /  \
          /    \
         /      \
    Integer Real
```

Problem 1.

Identify the following objects: (atom, list, none):

```
ATOM
(THIS IS AN ATOM)
(THIS IS A LIST)
((A B) (C D))
3
(3)
(LIST 3)
(/ (+ 3 1) (- 3 1))
)(
((()))
(() ())
((())
()))( ((A B C
```

Problem 2.

Evaluate the following:

```
(/ (+ 3 1) (- 3 1))
(* (MAX 3 4 5) (MIN 3 4 5))
(MIN (MAX 3 1 4) (MAX 2 7 1))
```

The primitives CAR and CDR

`(CAR L)` - returns the first element of the list L
`(CDR L)` - returns the list formed by the rest of the elements

Considering the expression: `(A B C)`
`(CAR '(A B C))`
returns
`A`
Notă: We notice that: `'(A B C) == (QUOTE (A B C))` and the function `QUOTE` returns the parameter of the function.
Whereas `(CDR '(A B C))` returns `(B C)`.

CAR şi CDR

```
(CAR '((A B) C))
(A B)
(CDR '((A B) C))
(C)
```

```
                  -------
          ----->| CAR |----> A ---------
          |        -------              |        --------
(A B C) ==|                             |==>| CONS |--> (A B C)
          |        -------              |        --------
```

```
                  ----->| CDR |----> (B C) -----
                          -------
```

The second element of the list is obtained this way:

```
(CAR (CDR '(A B C)))
B
```

Problem 3.

Evaluate the following:

```
(CAR '(P H W))
(CDR '(B K P H))
(CAR '((A B) (C D)))
(CDR '((A B) (C D)))
(CAR (CDR '((A B) (C D))))
(CDR (CAR '((A B) (C D))))
(CDR (CAR (CDR '((A B) (C D)))))
(CAR (CDR (CAR '((A B) (C D)))))
```

Problem 4.

Evaluate the following and justify the results obtained:

```
(CAR (CDR (CAR (CDR '((A B) (C D) (E F))))))
(CAR (CAR (CDR (CDR '((A B) (C D) (E F))))))
(CAR (CAR (CDR '(CDR ((A B) (C D) (E F))))))
(CAR (CAR '(CDR (CDR ((A B) (C D) (E F))))))
(CAR '(CAR (CDR (CDR ((A B) (C D) (E F))))))
'(CAR (CAR (CDR (CDR ((A B) (C D) (E F))))))
```

Problem 5.

Write down the CAR and CDR sequences that take out the C elements out of the following statements:

```
(A B C D)
((A B) (C D))
(((A) (B) (C) (D)))
(A (B) ((C)) (((D))))
((((A))) ((B)) (C) D)
((((A) B) C) D)
```

We can use compound primitives having the form CXXR, CXXXR, where X can be A (for CAR) or D (for CDR).

```
(CADR '(A B C)) == (CAR (CDR '(A B C)))
```

The symbolical atoms can have values A value can be attributed to a symbolical atom (number or symbol or list) at which that atom will be evaluated at. This is realized by the SETQ procedure(by side effect!).

```
(SETQ L '(A B))
L
(A B)
'L
L
(CAR L)
A
(CDR L)
(B)
```

APPEND, LIST and CONS procedures

APPEND unifies lists.

APPEND

```
(SETQ L '(A B))
(A B)
L
(A B)
(APPEND L L)
(A B A B)
```

```
(APPEND '(A) '() '(B) '())
(A B)
```

LIST constructs a list out of its arguments:

LIST

```
(LIST L L)
((A B) (A B))
(LIST 'L L)
(L (A B))
```

CONS inserts a new first element in a list:

CONS

```
(CONS 'A '(B C))
(A B C)
(CONS (CAR L) (CDR L)) ; == L
(A B)
```

```
                  -------
           ----->| CAR |----> A ---------
           |      -------                |      --------
(A B C) ==|                              |==>| CONS |--> (A B C)
           |      -------                |      --------
           ----->| CDR |----> (B C) -----
                  -------
```

APPEND, LIST, CONS

Follow these evaluations:

```
(APPEND '(A B) '(C D))
(A B C D)
(LIST '(A B) '(C D))
((A B) (C D))
(CONS '(A B) '(C D))
((A B) C D)
(APPEND L L)
(A B A B)
(LIST L L)
((A B) (A B))
(CONS L L)
((A B) A B)
(APPEND 'L L)
Error ('L is not a list)
(LIST 'L L)
(L (A B))
(CONS 'L L)
(L A B)
```

Problem 6.

Evaluate the statements:

```
(APPEND '(A B C) '())
(LIST '(A B C) '())
(CONS '(A B C) '())
```

LENGTH, REVERSE, SUBST and LAST procedures

LENGTH returns the length of a list.

LENGTH

```
(LENGTH '(A B))
2
(LENGTH '((A B) (C D)))
2
```

```
(LENGTH L)
2
(LENGTH (APPEND L L))
4
```

REVERSE inverts a list:

REVERSE

```
(REVERSE '(A B))
(B A)
(REVERSE '((A B) (C D)))
((C D) (A B))
```

SUBST replaces an expression with another in a list:
`(SUBST <new expr> <old expr> <list>)`

SUBST

```
(SUBST 'A 'B '(A B C))
(A A C)
```

LAST returns a list that contains the last element:

LAST

```
(LAST '(A B C))
(C)
(LAST '((A B) (C D)))
((C D))
```

Shapes evaluation

When the LISP interpreter receives a shape, it sends it to the primitive EVAL `EVAL`, which evaluates all arguments then calls the procedure from the head of the list.

```
 --------------------    Yes  -------------------------  Yes  ----------------------------------
< Is S an atom? >----->< Is S a number? >--------->| Return the number. |
 --------------------          --------------------------        ----------------------------------
        |                       |
        |                       | No
        | No                   V
        |               -------------------------------
        |               | Ret. value of S. |
        |               -------------------------------
       V
  ------------------------------              ------------------------------
< Is QUOTE the first >     Yes    | Return the second |
< atom of S?           >----------->| element of S.        |
 -------------------------------              -------------------------------
        |
        | No
       V
 -------------------------              -----------------------
< Does the first element      >    Yes     | Does not evaluate   |
< of S indicate a >---------->| arguments   |
< special behavior? >           | - special case. |
 --------------------------             -----------------------
       |
       | No
      V
 -----------------------------------------
| Uses EVAL on all   |
| elements of S except          |
| for the first             |
------------------------------------------
              |
              |
              V
 ------------------------------------
| Applies the first element  |
| of S to the resulted values          |
| and returns |
| the result.                  |
------------------------------------
```

Examples of evaluation

```
(SETQ ZERO 0 ONE 1 TWO 2 THREE 3 FOUR 4 FIVE 5 SIX 6 SEVEN 7 EIGHT 8
NINE 9)
9
ZERO
0
EIGHT
8
(SETQ A 'B)
B
(SETQ B 'C)
C
A
B
B
C
(EVAL A)
C
```

The `EVAL` procedure determines another evaluation of the arguments.

Problems

Problem 1. Atoms and lists.
Problem 2. Symbolical atoms. Functions.
Problem 3. CAR and CDR 1.
Problem 4. CAR and CDR 2.
Problem 5. CAR and CDR 3.
Problem 6. APPEND and LIST
Problem 7. Evaluate examples from LENGTH, REVERSE, SUBST, LAST and CONS
Problema 8. Evaluate examples from Shape evaluation