ACSL Training: Lisp

Sanjit Bhat Alexander Sun

January 13, 2019

1 Fun Facts

- Lisp was developed in the 50's by John McCarthy at MIT. He was a highly influential figure in Artificial Intelligence who later went on to win the Turing Award and found the Stanford AI Lab. Owing to its founder, Lisp was predominently used in AI research.
- After Fortran, Lisp is the second-oldest high-level programming language.
- Ever heard of tree data structures, dynamic typing (not specifying variable types), conditionals, or recursion? Yup, these concepts were pioneered in Lisp.

2 Background Info

First, I recommend spending some time on the Wikipedia page to get a background on Lisp-specific syntax. It delves into why certain things are the way they are.[1]

Next, check out the ACSL wiki for Lisp. Like the name implies, the fundamental concept in Lisp is that every line of code, from function calls to print statements, is placed inside a list. For example, (function arg1 arg2 arg3 ... argn) calls the function with the subsequent arguments as its input. The highly regular parentheses-based syntax makes the language easy to process. In fact, owing to the regularity, it was one of the first languages that allowed recursion—each of the arguments can be functions themselves.

Here's the relevant terminology:

1. *atom*: an individual list element that is not a list itself
2. *literal*: **SUPER IMPORTANT**. Defined with a single leading quote. Tells Lisp whether or not to evaluate a statement. Watch for the quotes whenever you read a problem.
3. *NIL*: Both an atom and a list. The equivalent of null in Java and NONE in Python.

2.1 Basic Functions

There are 4 basic functions: SET, SETQ, EVAL, and ATOM. SET assigns arg2 to arg1, evaluating the arguments as usual **if they are non-literals**. SETQ is the exact same as SET except that it automatically makes arg1 a literal.

[1] Did you know that Lisp uses prefix notation in its S-expressions? That's why prefix/postfix/infix is an ACSL topic

EVAL resolves arguments, while ATOM tests whether an item is an atom or a list. To test your knowledge, which of the two ATOMs will return "true"? What will (EVAL c) return:

1. (SETQ a '(MULT 2 3))
2. (SET 'b 'a)
3. (SET 'c a)
4. (ATOM b)
5. (ATOM c)

2.2 List Functions

There are 4 main list functions: CAR, CDR, CONS, REVERSE. (CAR x) returns the first element of list x and (CDR x) returns x without its first element. For quick access to specific elements in a list, Lisp users use functions like CAADDDAR where the A's and D's represent successive CAR and CDR operations. The preceding functions represents (CAR(CAR(CAD(CAD(CAD(CAR x)))))).

CONS(x y) is slightly more complex. It requires y to be a list and returns a list (x y). Finally, (REVERSE arg1 arg2 ... argn) returns (argn ... arg2 arg1).

What value does this code produce:

1. (SETQ z (CONS '(red white blue) (CDR '(THIS is a list))))
2. (CDDAR z)

2.3 Arithmetic Functions

There are 9 main arithmetic functions: ADD, SUB, MULT, DIV, SQUARE, EXP, EQ, POS, NEG. ADD and MULT work on an indefinite number of arguments, EQ tests for equality (returning "true" or NIL), and POS and NEG test the integer signs.

Note that ADD, SUB, MULT, and DIV can be written with +, -, *, and /, but remember to use **prefix** notation.

2.4 Defining Your Own Function

You can define your own function using the DEF function. For example, (DEF second (args) (CAR (CDR args))) creates a function second that returns its second argument.

3 Exercises

1. Evaluate (MULT (ADD 6 5 0) (MULT 5 1 2 2) (DIV 9 (SUB 2 5)))

2. Evaluate (CDR '((2 (3))(4 (5 6) 7)))

3. (SETQ X '(RI VA FL CA TX))
   (CAR (CDR (REVERSE X)))
   What is the value of the CAR expression?

4. Evaluate (CAR '((2 (3)) (4 (5 6)) 7))

5. (SETQ X '(((a (b c) d) e) ((b (c (d e) b)) a)(a b c)((c d) b (a b)(c e d))))
   Evaluate the following expression:
   (CDR(CDR(CDR(REVERSE(CAR(REVERSE(CDR x)))))))

6. Given the function definitions for HY and FY as follows:
   (DEF HY(PARMS)(REVERSE (CDR PARMS)))
   (DEF FY(PARMS)(CAR (HY(CDR PARMS))))
   What is the value of the following? (FY '(DO RE(MI FA) SO))

7. Evaluate (EXP(MULT 2(SUB 5(DIV(ADD 5 3 4) 2)) 3)3)

4 Solutions

1. -440
2. ((4 (5 6) 7))
3. CA
4. (2 (3))
5. (e d)
6. SO
7. -216