LISP Programming

LISP is one of the simplest computer languages in terms of syntax and semantics, and also one of the most powerful. It was developed in the mid-1950's by John McCarthy at M.I.T. as a "**LIS**t **P**rocessing language". Today, it is used for virtually all Artificial Intelligence programs and is the environment of choice for applications which require a powerful interactive working environment. LISP presents a very different way to think about programming from the "algorithmic" languages, such as BASIC, Fortran and Pascal.

As its name implies, the basis of LISP is a list. One constructs a list by enumerating elements inside a pair of parentheses. For example, here is a list with four elements (the second element is also a list):

(23 (this is easy) hello 821)

The elements in the list, which are not lists, are called "atoms." For example, the atoms in the list above are: 23, this, hello, 821, easy, and is. Everything in LISP is either an atom or a list (but not both). The only exception is "NIL," which is both an atom and a list. It can also be written as "()" – a pair of parentheses with nothing inside.

All statements in LISP are function calls with the following syntax: (*function* arg_1 arg_2 arg_3 ... arg_n). To evaluate a LISP statement, each of the arguments (possibly functions themselves) are evaluated, and then the function is invoked with the arguments. For example, (MULT (ADD 2 3) (ADD 1 4 2)) has a value of 35, since (ADD 2 3) has a value of 5, (ADD 1 4 2) has a value of 7, and (MULT 5 7) has a value of 35. Some functions have an arbitrary number of arguments; others require a fixed number. All statements return a value, which is either an atom or a list.

We may assign values to variables using the function SET. For example, the statement (SET 'test '6) would have a value of a 6, and (more importantly, however) would also cause the atom "test" to be bound to the atom "6". The quote in front of the arguments indicates that the arguments should not be evaluated before the function is invoked. The quote in front of numbers is optional. Observe the following examples:

Statement	Value	Comment
(SET 'a (MULT 2 3))	6	*a is an atom with a vaue of 6*
(SET 'a '(MULT 2 3))	(MULT 2 3)	*a is a list with 3 elements*
(SET 'b 'a)	a	*b is an atom with a value of the character a*
(SET 'c a)	(MULT 2 3)	*c is a list with 3 elements*
(SET 'TEST (ADD 3 (MULT 2 5)))	13	*test has a value of 13*
(SETQ VOWELS '(A E I O U))	(A E I O U)	*VOWELS is a list of 5 elements*
(SETQ x (SETQ y 'same))	same	*both x and y are bound to value of "same"*

(SETQ y 'diff)	diff	*x is still "same"*
(SETQ x y)	diff	*x is now "diff"*

The function SETQ is the same as SET, but it causes LISP to act as if the first argument was quoted. The function EVAL returns the value of its argument, after it has been evaluated. For example, (SETQ z '(ADD 2 3)) has a value of the list (ADD 2 3); the function (EVAL 'z) has a value of (ADD 2 3); the function of (EVAL z) has a value of 5 (but the binding of the atom z has not changed). In this last example, you can think of z being "resolved" twice: once because it is an argument to a function and LISP evaluates all arguments to functions before the function is invoked, and once when the function EVAL is invoked to resolve arguments. The function ATOM can be used to tell whether an item is an atom or a list.

Statement	Value	Comment
(SETQ X (ADD 45 8))	53	*X is now 53*
(SETQ Y '(ADD X 11))	(ADD X 11)	*Y is now a list*
(EVAL 'Y)	(ADD X 11)	
(EVAL Y)	64	
(ATOM 5)	true	
(ATOM X)	true	
(ATOM 'Y)	true	
(ATOM Y)	NIL	
(ATOM '(8 7 6))	NIL	

The two most famous LISP functions are CAR and CDR (pronounced: could-er), named after registers of a now long-forgotten IBM machine on which LISP was first developed. The function (CAR x) returns the first item of the list *x* (and *x* must be a list or an error will occur); (CDR x) returns the list without its first element (again, *x* must be a list). The function CONS takes two arguments, of which the second must be a list. It returns a list which is composed by placing the first argument as the first element in the second argument's list. The function REVERSE returns a list which is its arguments in reverse order. The following examples illustrate the use of CAR, CDR, and CONS:

Statement	Value
(REVERSE '(8 7 6))	(6 7 8)
(REVERSE '(1 (2 3 4) 5 6))	(6 5 (2 3 4) 1)
(CAR '(This is a list))	This
(CAR (CDR '(This is a list)))	is
(SETQ x (CDR (CDR '(a b c d))))	(c d)
(CDR '(This is a list))	(is a list)
(CAR '(hi))	hi
(CDR '(one))	NIL
(CAR '((1) (2 3) (4 5 6)))	(1)
(CONS 32 '(22 12))	(32 22 12)
(CONS 'first '(last))	(first last)

(CONS '(one) '(two three))	((one) two three)
(CDR (CAR '((red white) blue)))	(white)
(CONS (CAR '(red green blue)) (CDR '(brown tan gold)))	(red tan gold)
(CAR 'hi)	error
(CAR ())	error
(CAR '(ADD 2 2))	ADD
(CONS 78 NIL)	(78)
(CDR '((red green) (blue white))	((blue white))
(SETQ a '(orange black))	(orange black)
(ATOM (SETQ b (CONS 'red a)))	NIL
(ATOM b)	NIL
(ATOM (CAR b))	true
(ATOM NIL)	true
(SETQ fruit '(apple orange))	(apple orange)
(SETQ more (CONS 'grape fruit))	(grape apple orange)
(CDR (CDR (REVERSE more)))	(grape)
(SETQ colors '(red green yellow))	(red green yellow)
(SETQ area (MULT 2 3))	6
(SETQ A (CONS area '(CDR colors)))	(6 CDR colors)
(SETQ B (CONS 'area (CDR colors)))	(area green yellow)
(SETQ C (REVERSE colors))	(yellow green red)
(CAR (CDR C))	green
(CONS 'pink (CONS 'orange (CONS C NIL)))	(pink orange (yellow green red))

As you have probably deduced, the function ADD simply summed its arguments. We'll also be using the following arithmetic functions:

FUNCTION	RESULT
(ADD x_1 x_2 …)	sum of all arguments
(MULT x_1 x_2 …)	product of all arguments
(SUB *a b*)	*a-b*
(DIV *a b*)	*a*/*b*
(SQUARE *a*)	*a*a*
(EXP *a n*)	a^n
(EQ *a b*)	true if *a* and *b* are equal, NIL otherwise
(POS *a*)	true if *a* is positive, NIL otherwise
(NEG *a*)	true if *a* is negative, NIL otherwise

Some examples of these functions are as follows:

STATEMENT	VALUE
(ADD (EXP 2 3) (SUB 4 1) (DIV 54 4))	24.5
(SUB (MULT 3 2) (SUB 12 (ADD 2 2)))	-2
(ADD (SQUARE 3) (SQUARE 4))	25

LISP also allows us to create our own functions using the DEF function. For example,

```
(DEF SECOND(parms) (CAR (CDR parms)))
```

defines a new function called SECOND which operates on a single parameter named "parms". SECOND will take the CDR of the parameter and then the CAR of that result. So, for example:

```
(SECOND '(a b c d e))
```

would first CDR the list (yielding (b c d e)) and then CAR the result. So the value would be the single character "b". Consider the following program fragment:

```
(SETQ X '(a c s l))
(DEF WHAT(parms) (CONS parms (REVERSE (CDR parms))))
(DEF SECOND(parms) (CONS (CAR (CDR parms)) NIL))
```

The following chart illustrates the use of the user-defined functions WHAT and SECOND:

STATEMENT	VALUE
(WHAT X)	((a c s l) l s c)
(SECOND X)	(c)
(SECOND (WHAT X))	(l)
(WHAT (SECOND X))	((c))

Questions in this round will typically present a line of LISP code or a short sequence of statements and ask what is the value of the (final) statement.

References

Hofstadter, Douglas R. "Metamagical Themas," in *Scientific American*, February, 1983. Also, see the March and April, 1983 columns for a more detailed look. These articles can also be found in Hofstadter's delightful book, *Metamagical Themas* published by Basic Books (1985).

Charniak, Eugene and Drew McDermott. *An Introduction to Artificial Intelligence*, Addison-Welsley (1985).

Sample Problems

Evaluate: (CDR '((2 (3))(4 (5 6) 7)))	The CDR function takes the first element of its parameter (which is assumed to be a list) and returns the modified list. The first element of the list: ((2 (3))(4 (5 6) 7)) is (2 (3)), and the list without this element is ((4 (5 6) 7)).
Consider the following program fragment: (SETQ X '(RI VA FL CA TX)) (CAR (CDR (REVERSE X))) What is the value of the CAR expression?	The first statement binds variable *X* to the list (RI VA FL CA TX). The REVERSE of this list is the list (TX CA FL VA RI). whose CDR is (CA FL VA RI) The CAR of this list is just the atom "CA" (without the quotes).
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))	To evaluate (FY '(DO RE (MI FA) SO)), we must first evaluate (CAR (HY (CDR '(DO RE (MI FA) SO)))) Thus, HY is invoked with PARMS= '(RE (MI FA) SO), and we evaluate (REVERSE (CDR '(RE (MI FA) SO))) This has a value of (SO (MI FA)) which is returned to FY. FY now takes the CAR of this.
Evaluate the following expression. (EXP (MULT 2 (SUB 5 (DIV (ADD 5 3 4) 2)) 3) 3)	(EXP (MULT 2 (SUB 5 (DIV (ADD 5 3 4) 2)) 3) 3) (EXP (MULT 2 (SUB 5 (DIV 12 2)) 3) 3) (EXP (MULT 2 (SUB 5 6) 3) 3) (EXP (MULT 2-1 3) 3) (EXP –6 3) -216