LISP - LISTS

Lists had been the most important and the primary composite data structure in traditional LISP. Present day's Common LISP provides other data structures like, vector, hash table, classes or structures.

Lists are single linked lists. In LISP, lists are constructed as a chain of a simple record structure named **cons** linked together.

The cons Record Structure

A **cons** is a record structure containing two components called the **car** and the **cdr.**

Cons cells or cons are objects are pairs of values that are created using the function **cons.**

The **cons** function takes two arguments and returns a new cons cell containing the two values. These values can be references to any kind of object.

If the second value is not nil, or another cons cell, then the values are printed as a dotted pair enclosed by parentheses.

The two values in a cons cell are called the **car** and the **cdr.** The **car** function is used to access the first value and the **cdr** function is used to access the second value.

Example

Create a new source code file named main.lisp and type the following code in it.

```
(write (cons 1 2))
(terpri)
(write (cons 'a 'b))
(terpri)
(write (cons 1 nil))
(terpri)
(write (cons 1 (cons 2 nil)))
(terpri)
(write (cons 1 (cons 2 (cons 3 nil))))
(terpri)
(write (cons 'a (cons 'b (cons 'c nil))))
(terpri)
(write ( car (cons 'a (cons 'b (cons 'c nil)))))
(terpri)
(write ( cdr (cons 'a (cons 'b (cons 'c nil)))))
```

When you execute the code, it returns the following result:

```
(1 . 2)
(A . B)
(1)
(1 2)
(1 2 3)
(A B C)
A
(B C)
```

The above example shows how the cons structures could be used to create a single linked list, e.g., the list *ABC* consists of three cons cells linked together by their *cdrs.*

Diagrammatically, it could be expressed as:

Lists in LISP

Although cons cells can be used to create lists, however, constructing a list out of nested **cons** function calls can't be the best solution. The **list** function is rather used for creating lists in LISP.

The list function can take any number of arguments and as it is a function, it evaluates its arguments.

The **first** and **rest** functions give the first element and the rest part of a list. The following examples demonstrate the concepts.

Example 1

Create a new source code file named main.lisp and type the following code in it.

```
(write (list 1 2))
(terpri)
(write (list 'a 'b))
(terpri)
(write (list 1 nil))
(terpri)
(write (list 1 2 3))
(terpri)
(write (list 'a 'b 'c))
(terpri)
(write (list 3 4 'a (car '(b . c)) (* 4 -2)))
(terpri)
(write (list (list 'a 'b) (list 'c 'd 'e)))
```

When you execute the code, it returns the following result:

```
(1 2)
(A B)
(1 NIL)
(1 2 3)
(A B C)
(3 4 A B -8)
((A B) (C D E))
```

Example 2

Create a new source code file named main.lisp and type the following code in it.

```
(defun my-library (title author rating availability)
   (list :title title :author author :rating rating :availabilty availability)
)
(write (getf (my-library "Hunger Game" "Collins" 9 t) :title))
```

When you execute the code, it returns the following result:

```
"Hunger Game"
```

List Manipulating Functions

The following table provides some commonly used list manipulating functions.

Function	Description
car	It takes a list as argument, and returns its first element.
cdr	It takes a list as argument, and returns a list without the first element.
cons	It takes two arguments, an element and a list and returns a list with the element inserted at the first place.

list	It takes any number of arguments and returns a list with the arguments as member elements of the list.
append	It merges two or more list into one.
last	It takes a list and returns a list containing the last element.
member	It takes two arguments of which the second must be a list, if the first argument is a member of the second argument, and then it returns the remainder of the list beginning with the first argument.
reverse	It takes a list and returns a list with the top elements in reverse order.

Please note that all sequence functions are applicable to lists.

Example 3

Create a new source code file named main.lisp and type the following code in it.

```
(write (car '(a b c d e f)))
(terpri)
(write (cdr '(a b c d e f)))
(terpri)
(write (cons 'a '(b c)))
(terpri)
(write (list 'a '(b c) '(e f)))
(terpri)
(write (append '(b c) '(e f) '(p q) '() '(g)))
(terpri)
(write (last '(a b c d (e f))))
(terpri)
(write (reverse '(a b c d (e f))))
```

When you execute the code, it returns the following result:

```
A
(B C D E F)
(A B C)
(A (B C) (E F))
(B C E F P Q G)
((E F))
((E F) D C B A)
```

Concatenation of car and cdr Functions

The **car** and **cdr** functions and their combination allows extracting any particular element/ member of a list.

However, sequences of car and cdr functions could be abbreviated by concatenating the letter a for car and d for cdr within the letters c and r.

For example we can write cadadr to abbreviate the sequence of function calls - car cdr car cdr.

Thus, $cadadr'(a(cd\ efg))$ will return d

Example 4

Create a new source code file named main.lisp and type the following code in it.

```
(write (cadadr '(a (c d) (e f g))))
(terpri)
(write (caar (list (list 'a 'b) 'c)))
(terpri)
(write (cadr (list (list 1 2) (list 3 4))))
```

```
(terpri)
```

When you execute the code, it returns the following result:

```
D
A
(3 4)
```