DRONACHARYA GROUP OF INSTITUTIONS, GREATER NOIDA

Affiliated to Uttar Pradesh Technical University, Noida
Approved by AICTE

DEPARTMENT OF INFORMATION TECHNLOGY

Lab Manual for
Principles of Programming Language (ECS-553)

List of Experiments

1. Define a LISP function to compute sum of squares.
2. Define a LISP function to compute difference of squares. (if x > y return $x^2 - y^2$, otherwise $y^2 - x^2$).
3. Define a Recursive LISP function to solve Ackermann's Function.
4. Define a Recursive LISP function to compute factorial of a given number.
5. Define a Recursive LISP function which takes one argument as a list and returns last element of the list. (Do not use last predicate).
6. Define a Recursive LISP function which takes one argument as a list and returns a list except last element of the list. (Do not use but last predicate).
7. Define a Recursive LISP function which takes one argument as a list and returns reverse of the list. (Do not use reverse predicate).
8. Define a Recursive LISP function which takes two arguments first, an atom, second, a list, returns a list after.

LAB OBJECTIVE

- Overview of Rule Based Programming Language
- Basic Concept of Lisp Language
- Advance Programming

- Lisp is a family of computer programming languages with a long history and a distinctive, fully parenthesized Polish prefix notation.
- Lisp is the second-oldest high-level programming language in widespread use today
- Lisp was originally created as a practical mathematical notation for computer programs, influenced by the notation of Alonzo Church's lambda calculus. It quickly became the favored programming language for artificial intelligence (AI) research. As one of the earliest programming languages, Lisp pioneered many ideas in computer science, including tree data structures, automatic storage management, dynamic typing, conditionals, higher-order functions, recursion, and the self-hosting compiler.

LISP Connection to Artificial Intelligence:

- Lisp was closely connected with the artificial intelligence research community, especially on PDP-10 systems.
- Lisp was used as the implementation of the programming language Micro Planner which was used in the famous AI system SHRDLU.

Guidelines to Students

- Equipment in the lab for the use of student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage is caused is punishable.

- Students are required to carry their observation / programs book with completed exercises while entering the lab.

- Students are supposed to occupy the machines allotted to them and are not supposed to talk or make noise in the lab. The allocation is put up on the lab notice board.

- Lab can be used in free time / lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.

- Lab records need to be submitted on or before date of submission.

- Students are not supposed to use floppy disks

Experiment 1

Define a LISP function to compute sum of squares

```
(defun sumsqr (x y)
(+(* x x)(* y y)))

Write (sumsqr 2 3);
```

OUTPUT :

```
sumsqr

13
```

Experiment 2

Define a LISP function to compute differance of squares .(if x > y return x2– y2 , Otherwise y2 – x2)

```
(defun diffsqr (x y)
(if(> x y)
(-(* x x) (* y y))
(-(* y y) (* x x))))
```

Write (diffsqr 2 3)

OUTPUT :

DIFFSQR

5

Experiment 3

Define a Recursive LISP function to solve Ackermann's Function.

```
(defun ackermann (m n) "The Ackermann Function"
(cond ((= m 0) (+ n 1))
     ((= m 1) (+ n 2))
     ((= m 2) (+ 3 (* n 2)))
     ((= m 3) (+ 5 (* 8 (- (expt 2 n) 1))))
     (t (cond ((= n 0) (ackermann (- m 1) 1))
              (t (ackermann (- m 1) (ackermann m (- n 1))))
       ))
))
```

Write (ackermann 2 3)

OUTPUT :

ACKERMANN

9

Experiment 4

Define a Recursive LISP function to compute the factorial of given number.

```
(defun factorial (N)
  "Compute the factorial of N."
  (if (= N 1)
      1
    (* N (factorial (- N 1)))))

Write (factorial 5)
```

OUTPUT :

```
FACTORIAL

120
```

Experiment 5

Define a Recursive LISP function which takes one argument as a list and return last element of The list.(do not use last predicate.)

```
(defun last_element(ab_list)
(first(reverse ab_list)))


Write (last_element (a b c d))
```

OUTPUT :

```
LAST_ELEMENT

D
```

Experiment 6

Define a Recursive LISP function which takes one argument as a list and return list except last element of the list.(do not use butlast.)

```
(defun not_last(ab_list)
(reverse(rest(reverse ab_list))))
```

```
Write (not_last '(a b c d e))
```

OUTPUT :

```
NOT_LAST

(A B C D)
```

Experiment 7

Define a Recursive LISP function which takes one argument as a list and return reverse of the list. (do not use reverse predicate).

```
(defun list-append (L1 L2)
  "Append L1 by L2."
  (if (null L1)
      L2
    (cons (first L1) (list-append (rest L1) L2))))


(defun show-list-reverse (L)
  "Create a new list containing the elements of L in reversed order."
  (if (null L)
      nil
    (list-append (show-list-reverse (rest L))
                 (list (first L)))))
```

Write (show-list-reverse '(1 2 3 4))

OUTPUT :

LIST-APPEND

SHOW-LIST-REVERSE

(4 3 2 1)

Experiment 8

Define a Recursive LISP function which takes two argument first an atom second a list returns a list after removing first occurrence of that atom within the list.

```
(defun remove(lst elt)
(cond((null lst)nil)
((equal(first lst)elt)(rest lst))
(elt(cons(first lst)
(remove(rest lst)elt)))))
```

Write (remove '(1 2 3 3 4 4)'3)

OUTPUT :

REMOVE

(1 2 3 4 4)