Artificial Intelligence

Lecture 13

LISP (LISt Processing)

Prepared by:

Md. Mijanur Rahman, Prof. Dr.

Dept. of CSE, Jatiya Kabi Kazi Nazrul Islam University

Email: mijanjkkniu@gmail.com

LISP

Lecture Outlines:

- …
- Functions
- Conditions (cond / if)
- Arrays
- Input & Output

Functions in LISP…

- **Defining function:**
- The macro named **defun** is used for defining functions. The **defun** macro needs three arguments −
  - Name of the function
  - Parameters of the function
  - Body of the function

  Syntax for **defun** is −

  (**defun name (par1 par2 …) "*Optional documentation string*" body**)

Functions in LISP

- **Example 1**

Let's write a function named *averagenum* that will print the average of four numbers. We will send these numbers as parameters.

```
(defun averagenum (n1 n2 n3 n4)
   (/ ( + n1 n2 n3 n4) 4)
)
(write(averagenum 10 20 30 40))
```

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

```
25
```

- **Example 2**

Let's define and call a function that would calculate the area of a circle when the radius of the circle is given as an argument.

```
(defun area-circle(rad)
   "Calculates area of a circle with given radius"
   (terpri)
   (format t "Radius: ~5f" rad)
   (format t "~%Area: ~10f" (* 3.141592 rad rad))
)
(area-circle 10)
```

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

```
Radius:  10.0
Area:    314.1592
```

Conditions in LISP…

- Cond (for conditional) like if-then-else construct.
- The syntax for **cond** is:

(cond (<test1> <action1>)
(<test2> <action2>)
-
-
-
(<testk> <actionk>))

- Each (<$test_i$><$action_i$>), i =1,2,3,…, k, is a clause.

- **Example:** The following function returns the maximum of two numbers:

```
(defun max2 (a b)
      (cond   ((> a b) (setq x a))
              (t   (setq x b) )
       )
      (format t "Max: ~4d" x)
)
(max2 234  123)
```

Output:

Max: 234

Conditions in LISP…

- **If construct:**
- The **if** macro is followed by a test clause that evaluates to t or nil.
- If the test clause is evaluated to the t, then the action following the test clause is executed. If it is nil, then the next clause is evaluated.

- **Syntax:**

```
(If  (test-clause)  (action1)  (action2))
```

- **Example:** The following function returns the maximum of three numbers:

```
(defun max3 (a b c)
  (if (>= a b)
    (if (>= a c) (setq x a)
        (setq x c))
    (if (>= b c) (setq x b)
        (setq c)))
  (format t "Max: ~3d" x)
)
(max3 13 11 14)
```

Output:

Max: 14

Conditions in LISP

- **Recursion:**
- **Example:**

The following function returns the factorial of a number:

```
(defun fact (n)
 "Compute the factorial of N."
 (cond ((zerop n) 1)
   (t (* n( fact (- n 1))))
  )
)
(setq f (fact 6))
(format t "Factorial: ~4d" f)
```

- **Output:**

  Factorial: 720

- **Example:** The following function returns the Nth Fibonacci number:

```
(defun fibonacci (N)  "Compute
the N'th Fibonacci number."
(if (or (zerop N) (= N 1))  1
   (+ (fibonacci (- N 1))
      (fibonacci (- N 2)))))
(setq fib (fibonacci 6))
(format t "Nth Fibonacci:~4d"
fib)
```

- **Output:**

  Nth Fibonacci: 13

Loops in LISP…

- **For loop construct:**
- The for loop construct follows several syntax:

```
(loop for loop-var in <a list>
   do (action)
)

(loop for loop-var from val1 to val2
   do (action)
)
```

- **Example:**

```
(loop for x in '(Tom Dick
Harry)
   do (format t " ~s" x)
)
```

Output:

```
    Tom Dick Harry
```

```
(loop for i from 10 to 15
   do (print i)
)
```

Output:

10 11 12 13 14 15

Loops in LISP

- **Terminate the loop with a test (until, while)**

```
(loop for x in '(1 2 3 4 5)
   until (> x 3)
        do (format t "x is ~a~&" x)
   collect x)
_
(loop for x in '(1 2 3 4 5)
   while (< x 4)
        do (format t "x is ~a~&" x)
   collect x)
```

Do and collect can be combined in one expression.

- **Loop…repeat:**

```
(loop repeat 10
   do (format t "Hello!~%) )
;This prints 10 times Hello!
```

- **Sum and Loop:**

```
(loop for i from 1 to 3
   sum (* i i) into total
   do (print i)
   finally (print total))
```

Output:

```
1 2 3  14
```

LISP – Arrays…

- In LISP, an array element is specified by a sequence of non-negative integer indices. The length of the sequence must equal the rank of the array. Indexing starts from zero.
- For example, to create an array with 10- cells, named my-array, we can write −

  (setf my-array (male-array '(10)))
- The aref function allows accessing the contents of the cells. It takes two arguments, the name of the array and the index value. For example: (aref my-array 9)
  - **Code:**

```
(setf a (make-array '(10)))
(setf (aref a 0) 25)
(setf (aref a 1) 23)
(setf (aref a 2) 45)
(setf (aref a 3) 10)
(setf (aref a 4) 20)
(setf (aref a 5) 17)
(setf (aref a 6) 25)
(setf (aref a 7) 19)
(setf (aref a 8) 67)
(setf (aref a 9) 30)
(write a)
```

  - **Output:**

(25 23 45 10 20 17 25 19 67 30)

  - **3-by-3 array**

```
(setf x (make-array '(3 3)
:initial-contents
 '((0 1 2 ) (3 4 5) (6 7 8))))
(write x)
```

Output:

#2A ((0 1 2) (3 4 5) (6 7 8))

Accessing Elements:

(aref x 1 0) returns 3

(aref x 0 2) returns 2

LISP - Arrays

- To loop over an array nested loops can be used
- **For example:**

```
(defparameter a #2A((1 2 3) (4 5 6) (7 8 9)))
(destructuring-bind (n m) (array-dimensions a)
    (loop for i from 0 below n do
        (loop for j from 0 below m do
            (format t "a[~a ~a] = ~a~%" i j (aref a i j)))))
```

- **Output:**

a[0 0] = 1

:

A[2 2] = 9

LISP - Input & Output…

- **Reading Input from Keyboard**

The **read** function is used for taking input from the keyboard. It may not take any argument.

For example, consider the code snippet –

```
(write ( + 15.0 (read)))
```

Assume the user enters 10.2 from the STDIN Input, it returns,

```
25.2
```

- **Code:**

(setq x (read))

(write (+ 15.0 x))

LISP - Input & Output…

- **Reading Input from Keyboard**
- **Example:**

```
; calculates area of a circle
; when the radius is input from keyboard
(defun AreaOfCircle()
(terpri)
(princ "Enter Radius: ")
(setq radius (read))
(setq area (* 3.1416 radius radius))
(princ "Area: ")
(write area))
(AreaOfCircle)
```

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

```
Enter Radius: 5 (STDIN Input)
Area: 78.53999
```

LISP - Input & Output…

- **Reading Input from Keyboard**
- **Example: Multiple Input**

```
; Multiple numbers from keyboard
(defun numSum()
(terpri)
(princ "Enter numbers: ")
(setq n1 (read))(setq n2 (read))(setq n3 (read))
(setq sum (+ n1 n2 n3))
(princ "Sum: ")
(write sum))
(numSum)
```

Output:

```
Enter numbers: 3 4 5 (STDIN Input)
Sum: 12
```

LISP - Input & Output…

- **The Output Functions**

Example

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

```
; this program inputs a numbers and doubles it
(defun DoubleNumber()
   (terpri)
   (princ "Enter Number : ")
   (setq n1 (read))
   (setq doubled (* 2.0 n1))
   (princ "The Number: ")
   (write n1)
   (terpri)
   (princ "The Number Doubled: ")
   (write doubled)
)
(DoubleNumber)
```

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

```
Enter Number : 3456.78 (STDIN Input)
The Number: 3456.78
The Number Doubled: 6913.56
```

LISP - Input & Output

- **Formatted Output**

Example

Let us rewrite the program calculating a circle's area −

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

```
(defun AreaOfCircle()
   (terpri)
   (princ "Enter Radius: ")
   (setq radius (read))
   (setq area (* 3.1416 radius radius))
   (format t "Radius: = ~F~% Area = ~F" radius area)
)
(AreaOfCircle)
```

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

```
Enter Radius: 10.234 (STDIN Input)
Radius: = 10.234
Area = 329.03473
```

LISP (List Processing)

THE END