Artificial Intelligence

Lecture 12

LISP (LISt Processing)

Prepared by:

Md. Mijanur Rahman, Prof. Dr.

Dept. of CSE, Jatiya Kabi Kazi Nazrul Islam University

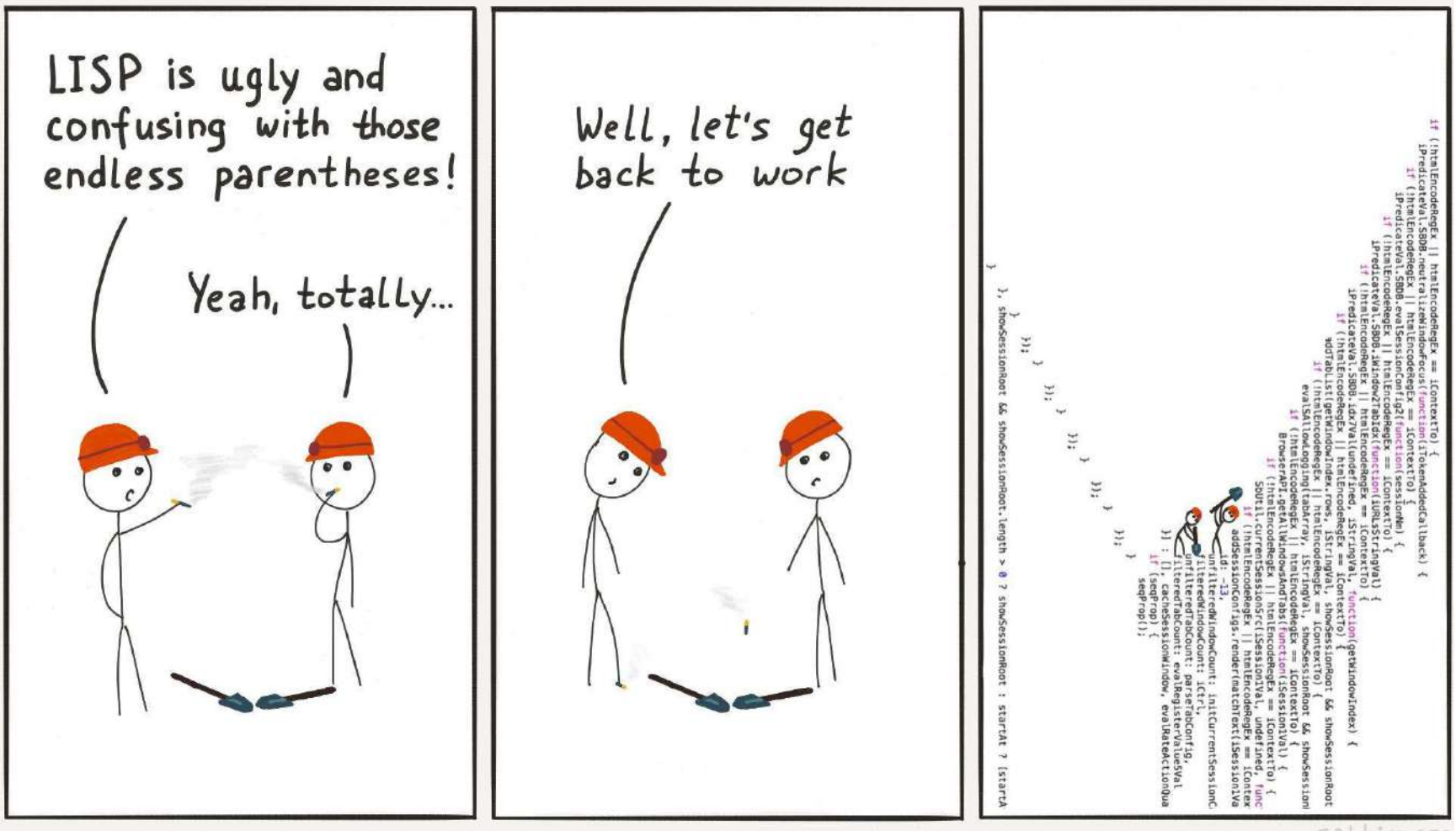
Email: mijanjkkniu@gmail.com

LISP

Lecture Outlines:

- LISP – History
- LISP – Environment
- LISP - Program Structure
- Syntax of LISP Program
- LISP – Variables
- Numeric Functions in LISP
- List Manipulation Functions in LISP
- Predicates in LISP
- …

LISP - History…



LISP - History

- Lisp (historically LISP) is a family of programming languages with a long history and a distinctive, fully parenthesized prefix notation.
- John McCarthy invented LISP in 1958, shortly after the development of FORTRAN, while he was at MIT. It was first implemented by Steve Russell on an IBM 704 computer. Lisp was originally created as a practical mathematical notation for computer programs.
- The name *LISP* derives from "LISt Processor“ or "LISt Processing". Linked lists are one of Lisp's major data structures, and Lisp source code is made of lists. Thus, Lisp programs can manipulate source code as a data structure.
- It is particularly suitable for Artificial Intelligence programs, as it processes symbolic information effectively.

LISP - Environment

- **Local Environment Setup**
  - If you are still willing to set up your environment for Lisp programming language, you need the following two software available on your computer:

    (a) Text Editor and (b) The Lisp Executer.

  - **Text Editor**
    - This will be used to type your program. Examples of few editors include Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, and vim or vi.
  - **The Lisp Executer**
    - The source code written in source file is the human readable source for your program. It needs to be "executed", to turn into machine language so that your CPU can actually execute the program as per instructions given.

LISP - Program Structure…

- LISP expressions are called **symbolic expressions or s-expressions**. The s-expressions are composed of three valid objects, atoms, lists and strings. Any s-expression is a **valid program**.
- LISP programs run either on **an interpreter or as compiled code**. It reads the program code, evaluates it, and prints the values returned by the program.
- **A Simple Program**
  - Let us write an s-expression to find the sum of three numbers 7, 9 and 11. To do this, we can type at the interpreter prompt.

    (+ 7 9 11)
  - LISP returns the result −

    27
  - If you would like to run the same program as a compiled code, then create a LISP source code file named **myprog.lisp** and type the following code in it.

    (write (+ 7 9 11))
  - When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result returned is −

    27

LISP - Program Structure

- **LISP Uses Prefix Notation**
  - You might have noted that LISP uses prefix notation.
  - **Example:** Let us write code for converting Fahrenheit temp of 60° F to the centigrade scale.

    The mathematical expression for this conversion will be:

    (60 * 9 / 5) + 32
  - Create a source code file named main.lisp and type the following code in it:

    (write (+ (* (/ 9 5) 60) 32)

    When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result returned is:

    140
- **The 'Hello World' Program**

```
(write-line "Hello World")
(write-line "I am learning LISP")
```

Result returned:

Hello World
I am learning LISP

- **LIVE Program**: https://www.tutorialspoint.com/execute_lisp_online.php
- Online coding platforms: https://www.tutorialspoint.com/codingground.htm

Evaluation of LISP Programs

- Evaluation of LISP programs has two parts −
  - Translation of program text into Lisp objects by a reader program
  - Implementation of the semantics of the language in terms of these objects by an evaluator program
- The evaluation process takes the following steps −
  - The reader translates the strings of characters to LISP objects or s-expressions.
  - The evaluator defines syntax of Lisp forms that are built from s-expressions. This second level of evaluation defines a syntax that determines which s-expressions are LISP forms.
  - The evaluator works as a function that takes a valid LISP form as an argument and returns a value. This is the reason why we put the LISP expression in parenthesis, because we are sending the entire expression/form to the evaluator as arguments.

LISP – Basic Syntax…

- The basic building blocks are:
  - Atom
    - An atom is a number or string of contiguous characters, including numbers and special characters.
  - List
    - A list is a sequence of atoms and/or other lists enclosed within parentheses.
  - String
    - A string is a group of characters enclosed in double quotation marks.

LISP – Basic Syntax…

- **Valid Atoms:**
  - This-is-a-symbolic-atom
  - Bill
  - 120002345
  - A12345
  - *var*
  - Block#5
- **Valid Lists**
  - (this a list)
  - (a b c d)
  - (a (b c) e f)
  - (father sam (joe bill sue)
- **Valid Strings**
  - “This is a string”
  - “Enter a number”
  - “Nazrul University”

```
(Lots of
 (Irritating
  (Superfluous
   (Parentheses))))
```

LISP – Basic Syntax…

- **Adding Comments**
  - The semicolon symbol (;) is used for indicating a comment line.

For Example:

```
; My first LISP Program
(write-line "Hello World")
(write-line "I am learning LISP")
```

LISP – Basic Syntax

- Following are some of the important points to note −
  - The basic numeric operations in LISP are +, -, *, and /
  - LISP represents a function call f(x) as (f x), for example cos(45) is written as cos 45
  - LISP expressions are case-insensitive, cos 45 or COS 45 are same.
  - LISP tries to evaluate everything, including the arguments of a function. Only three types of elements are constants and always return their own value
    - Numbers
    - The letter **t,** that stands for logical true.
    - The value **nil,** that stands for logical false, as well as an empty list.

Numeric Functions in LISP

- Predefined numeric functions:

Function Call	Value Returned
(+ 3 5 8 4)	20
(- 10 12)	-2
(* 2 3 4)	24
(/ 24 3)	8

- Command Prompt:

```
   5+6+9
-> (+ 5 6 9)
  20
->
```

```
50*9/5+32
    ->(+ (* (/ 9 5) 50) 32)
      122
->
```

LISP – Variables…

- **Global Variables**
  - Global variables have permanent values throughout the LISP system and remain in effect until a new value is specified.
  - Global variables are generally declared using the **defvar** construct. For example:

    (defvar x 112)

    (write x)
  - You can also use **setq** construct. For example:

```
(setq x 10)
(setq y 20)
(format t "x = ~2d y = ~2d ~%" x y)

(setq x 100)
(setq y 200)
(format t "x = ~2d y = ~2d" x y)
```

Result:

```
x = 10 y = 20
x = 100 y = 200
```

LISP – Variables

- **Local Variables**
- Local variables are defined within a given procedure. The parameters named as arguments within a function definition are also local variables. Local variables are accessible only within the respective function.
- Like the global variables, local variables can also be created using the **setq** construct. There are two other constructs - **let** and **prog** for creating local variables.
  - The let construct has the following syntax.

(let ((var1 val1) (var2 val2) …. (varn valn)) <s-expression>)

  - Example:

```
(let ((x 'a) (y 'b)(z 'c))
(format t "x = ~a y = ~a z = ~a" x y z))
```

```
x = A y = B z = C
```

```
(prog ((x '(a b c))(y '(1 2 3))(z '(p q 10)))
(format t "x = ~a y = ~a z = ~a" x y z))
```

```
x = (A B C) y = (1 2 3) z = (P Q 10)
```

List Manipulation Functions in LISP

- **Basic list manipulation functions:**

Function Call	Value Returned
(car '(a b c))	a
(cdr '(a b c))	(b c)
(cons 'a '(b c))	(a b c)
(list 'a '(b c))	(a (b c))

- cadadr – car cdr car cdr.

-> (cadadr '(a (b c) b))

c

- **Command Prompt:**

-> (cons '(* 2 3) '(1))

((* 2 3) 1)

-> (cons (* 2 3) '(1))

(6 1)

-> (car (cdr '(a b c d)))

b

-> (cons (car '(a b c) (cdr '(a b c)))

(a b c)

Predicates in LISP

- **Predicate Functions:**

atom	equal
listp	evenp
null	zerop
oddp	numberp
<=	greaterp (or >)
>=	lessp (or <)

Function Call	Value Returned
(atom ‘aabb)	t
(equal ‘a (car ‘(a b))	t
(lessp 5 2 1 3)	nil
(oddp 5)	t

List Processing

TO BE CONTINUED…