G22.2210

Programming Languages: PL

B. Mishra
New York University.

Lecture # 13

—Slide 1—

Functional Programming

- **Pure Functional Programming**:
  Implicit Principle
  ◦ The value of an expression depends only on the values of its subexpressions, if any.
  - No side-effect. (No State—No assignment)
  - An expression has the same value, every time.
  - Implicit Storage Management:
    Allocation on Demand + Garbage Collection.
  - Functions are **First Class Objects**:
    1) As value of an expression
    2) As parameters
    3) As data Objects.

—Slide 2—

Common Lisp

- LISP: LIst Processing Language
  Not— Lots of Insidious Sill Parentheses
- Second oldest Programming Language (After Fortran)
- Application Areas:
  1. Theorem Proving
  2. Symbolic Algebra
  3. AI (Artificial Intelligence)
     (Natural Language Processing, Computer Vision, Robot Control Systems, Expert Systems, Neural Networks, Automatic Programming)

—Slide 3—

HISTORY

- Developed at MIT AI Lab—1959.
  LISP 1.5 running on an IBM machine.
- BBN LISP (PDP 1/SDS 940) became → INTERLISP (PDP 10)
- MACLISP (MIT Project MAC)
- LISP 1.6—A version of MACLISP
  ◦ UCI-LISP (Univ. of Cal. at Irvine)
  ◦ Standard Lisp (Univ. of Utah)
- Lisp Machine Lisp
  Large Personal Lisp Machine built at MIT
- FranzLISP for Vax/UNIX (UC Berkeley)
- NIL for Vax/VMS (MIT)
- Scheme at MIT
- T Lisp at Yale

—Slide 4—

Common Lisp

- 1981/Carnegie-Mellon/Guy L. Steele
- **Clean Lisp**
  Inconsistencies and illogical conventions were resolved
- **Transportable**
  Programs written in Common Lisp and debugged in one implementation should run on another machine/implementation without change.

—Slide 5—

Lisp Data & Functional Objects

- Lisp Programs and the data have Same form
  - Self-modifying Lisp programs.
  - Embedded languages in Lisp
- Lisp functions are data objects that can be passed as parameters to other functions
  - Extensible control structures
- Interpreter + Compiler
- Garbage Collector

—Slide 6—

Lisp Objects

- **Lisp Objects**
  1. ATOMS
     (a) Numbers: $\{3, -5.7, 2010014567\}$
     (b) Symbols: { `A`, `EVAL`, `PI`, `T`, `NIL` }
  2. CONS
     Conjunctions of two Lisp objects. Each of them may be a CONS object or an ATOM.
- Example

```
(CONS 'A 'B)
(CONS '2 (CONS 'BAD 'NIL))
```


—Slide 7—

CONS Cells

=Two Compartments...
Each compartment holds an atom or a cons object (A pointer to a cons cell.

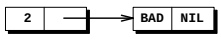
- **(CONS 'B 'NIL)**

(B)

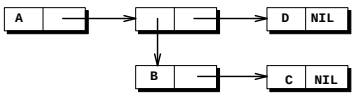
- **(CONS 'A 'B)**

(A . B)

- **(CONS '2 (CONS 'BAD 'NIL))** **(2 BAD)**



- **(CONS 'A (CONS**
  (CONS 'B (CONS 'C 'NIL))
  (CONS 'D NIL)
)



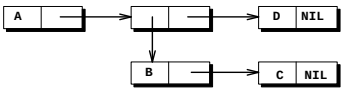
(A (B C) D)

—Slide 8—

LISTS

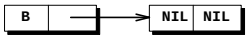
- Function `LIST`

```
   (LIST 'A (LIST 'B 'C) 'D)
=> (A (B C) D)
```



(A (B C) D)

```
   (LIST 'B NIL)
=> (B NIL)
```




```
   (LIST)
=> NIL
```

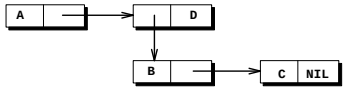
(An Atom not a `CONS` cell)

—Slide 9—

Proper and Improper Lists

- **Proper Lists**
  Lists terminating in NIL
- **Improper Lists**
  Lists not terminating in NIL

```
(A (B C) . D)
```



—Slide 10—

List Operations

- `CAR`: Extracts the *first* element of a list
- `CDR`: Extracts the *rest* (all but the first element) of a list
- `CAR` & `CDR` can be applied to any list, but not to atom other than `NIL`

$$(CAR\ NIL) \equiv (CDR\ \ NIL) \equiv NIL$$

- **Examples**

```
(CAR '(A B C))                          => A
(CDR '(A B C))                          => (B C)
(CAR (CDR (CAR (CDR '(A (B C) D)))))    => C
```


—Slide 11—

List Predicates

- Boolean-valued functions
  {`T` = True, `NIL` = False }
- `ATOM`: True iff an atom

```
(ATOM 'NIL)          =>   T
(ATOM '(X Y))        =>   NIL
```

- `CONSP`, `LISTP`, ...
- `NULL`: True iff an empty list (e.g., `NIL`)

```
(NULL 'NIL)          =>   T
(NULL 'X)            =>   NIL
```

- `ZEROP`, `NUMBERP`, ...

—Slide 12—

Conditional Form

- IF & COND

```
(IF <test-predicate> <then-clause> <else-clause>)
(COND (<test-1> <result-1>)
      (<test-2> <result-2>)
      ...
      (<test-n> <result-n>)
      )
```

- IF form is equivalent to

```
(COND (<test-predicate> <then-clause>)
      (T <else-clause>))
```

- When the LISP evaluation procedure sees a COND, it evaluates each of the test forms in order (i.e., <test-1>, <test-2>, ...
- When one evaluates to "non-NIL," its corresponding result form (i.e., <result-i>) is evaluated and its value is returned as the result of the COND.

—Slide 13—

Simple Examples

- **Example**

```
(IF (ATOM 'X) 'YES 'NO)              =>  YES

(COND ((ATOM 'X) 'YES)
      (T 'NO))                       =>  YES

(DEFUN LENGTH (LIST)
      (IF (NULL LIST)
          0
          (+ 1 (LENGTH (CDR LIST)))))
```

- LENGTH Computation

```
(LENGTH '())                         =>  0

(LENGTH '(C))
=  (+ 1 (LENGTH '()))                =>  1

(LENGTH '(B C))
=  (+ 1 (LENGTH '(C)))               =>  2

(LENGTH '(A B C))
=  (+ 1 (LENGTH '(B C)))             =>  3
```


—Slide 14—

LISP EVALUATOR: EVAL

LISP Obj ⇒ S-expression (Symbolic Expn)
↦ Values

Evaluation Order = "Inside-Out"

- Atoms evaluate to themselves:
  Numbers and NIL are constants and have predefined values. Symbols can be SET to a value in the course of execution.

```
(SET 'A '(X Y))                =>  A = (X Y)
```

- List to be evaluated must have a "functional-valued" symbol as its CAR. The CDR of the list is treated as the arguments.
  Evaluation proceeds by applying the function (CAR) to the argument (CDR)

```
(+ 2 4)                          =>  6
(+ (CAR (CDR (LIST 2 3))) 4)     =>  7
```


—Slide 15—

QUOTE

- Evaluation Blocker
  - If the symbol in the functional position is a *special form*, then the usual evaluation rule is replaced by a special one *before* the arguments are evaluated
  - QUOTE ≡ Special Form

A quoted item evaluates to itself

```
(QUOTE A)                  => 'A
(QUOTE (A B (QUOTE C) D))  => '(A B 'C D)
```

- **Example**

```
(SET 'PI 3.1415)           =>  PI = 3.1415
                           =>  'PI = PI
(SETQ B 'X)
(SET B 'Y)                 =>  B = X
                           =>  X = Y
(CONS 3 '(+ 5 6))          =>  (3 + 5 6)
(+ '3 '4)                  =>  7
```

—Slide 16—

EVAL

- Evaluation procedure can be invoked directly

```
   (SETQ A (LIST '+ 5 6))      =>  A = (+ 5 6)

   (CONS A '(IS MY RESULT))
=> ((+5 6) IS MY RESULT)

   (CONS (EVAL A) '(IS MY RESULT))
=> (11 IS MY RESULT)
```


—Slide 17—

USER-DEFINED FUNCTIONS

`(DEFUN <function-name> <argument-list> <body>)`

- `DEFUN`
  Special Form: subforms are not evaluated as arguments
- `<function-name>`
  Must be a symbol. Can be any symbol other than `NIL`
- `<argument-list>`
  Number of Symbols in the argument list is the number of arguments the function must be handed by the `EVAL` procedure
- `<body>`
  An S-expression that will be evaluated when the function is called... After appropriate substitution of the actuals for formals.

—Slide 18—

Examples: APPEND & REVERSE

```
(DEFUN APPEND (X Y)
   (IF (NULL X)
       Y
      (CONS (CAR X) (APPEND (CDR X) Y))))


(DEFUN REVERSE (LIST)
   (IF (NULL LIST)
       'NIL
       (APPEND (REVERSE (CDR LIST))
               (CONS (CAR LIST) 'NIL))))
```

[End of Lecture #13]