V22.0490.001
Special Topics: Programming Languages

B. Mishra

New York University.

Lecture # 11

—Slide 1—

Common Lisp
Language Survey 4
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—

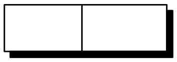
Lisp Objects

- **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))
```

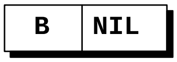
—Slide 7—

CONS Cells



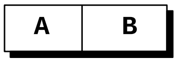
= =Two Compartments...
 Each compartment holds an atom
or a cons object (A pointer to a
cons cell.

- **(CONS ’B ’NIL)**



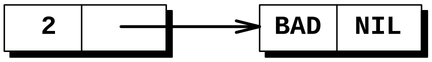
(B)

- **(CONS ’A ’B)**



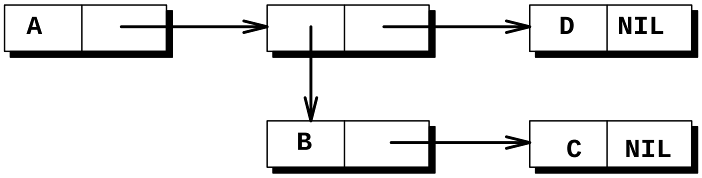
(A . B)

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



- **(CONS ’A (CONS**

```
(CONS ’A (CONS
            (CONS ’B (CONS ’C ’NIL))
            (CONS ’D NIL)
         )
```



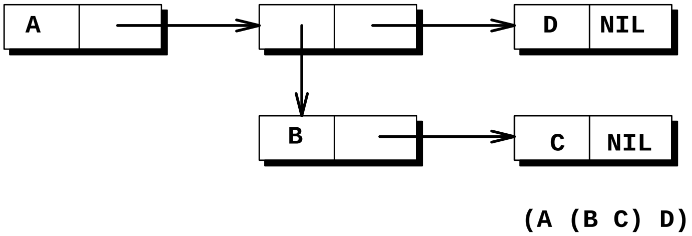
(A (B C) D)

—Slide 8—

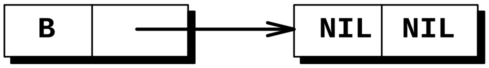
LISTS

- Function `LIST`

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



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



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

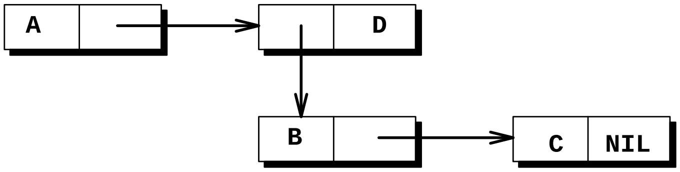
(An Atom not a `CONS` cell)

—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) ≡ (CDR NIL) ≡ NIL

- **Examples**

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

—Last Slide—

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`**, . . .

[End of Lecture #11]