V22.0490.001
Special Topics: Programming Languages

B. Mishra
New York University.

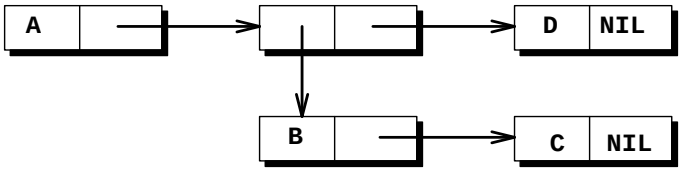
Lecture # 12

—Slide 1—

LISTS

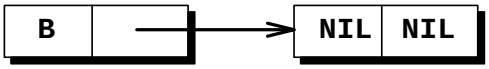
- Function `LIST`

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



(A (B C) D)

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




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

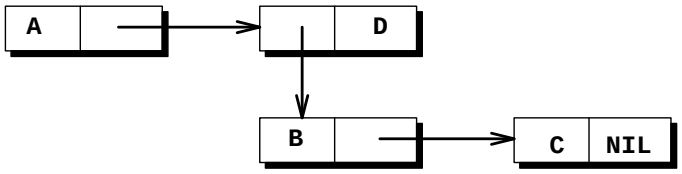
(An Atom not a `CONS` cell)

—Slide 2—

Proper and Improper Lists

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

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



—Slide 3—

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
```

—Slide 4—

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 5—

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 6—

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 7—

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 8—

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 9—

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 10—

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.

—Last Slide—

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 #12]