Lisp

- History:

  "**LIS**t **P**rocessing"

  Old language (1960), but still in use.

- Based on the "lambda calculus"

- Current Dialects of Lisp:

  Pure Lisp

  Franz Lisp (Berkeley)

  MacLisp (MIT), InterLisp (Xerox, BBN)

  Common Lisp

  (all are basically the same)

- Permits both

  functional and *procedural* programming

Lisp Logistics

- Use Levesque/Reiter/Hirst "Notes"
  from Chapter 2-5

- Dialect differences:
  "Notes" in *Pure* Lisp
  Actual code in *CommonLisp*
  other refs in *FranzLisp*
  [Use "LispCraft", "Common LispCraft" books]

- Get CONCEPTs first
  Many from Pencil/Paper exercises

Lisp is Interactive

E.g., on `obed` machine:

```
obed:~> gcl
GCL (GNU Common Lisp) Version(2.2) Tue May 13 14:32:42 MDT 1997
Licensed under GNU Public Library License
Contains Enhancements by W. Schelter
> (+ 11 23)
34
> (setq x (* 3 4))
12
> (+ (* x 2) 5)
29
> (defun sq (y) (* y y))
sq
> (sq x)
144
> ^D
obed:~>
```

Lisp's Data Structure

- Everything in *Lisp* represented by
  "symbolic expressions"
  (called *s-expressions*, or <u>*s-exprs*</u>)

- An *s-expr* is either

  – an *Atom*,   or

  – a List of s-exprs

- An *Atom* is
  a string of 1 or more letters or digits.
  E.g.,

g	u-of-t
24	cdr
1a2b	8044

N.B.: Numbers are atoms too.

Lists in Lisp

- Def'n

  A *list* is 0 or more s-exprs
  enclosed in parentheses.

- Examples

  (a b c)
  ()
  (+ 2 3)
  (plus x (times y 3))
  (() ())

- General: $(\mathtt{s}_1\ \mathtt{s}_2\ \cdots\ \mathtt{s}_n)$

  $n \geq 0$, $\mathtt{s}_i$ are s-exprs

- Special case: () is the *empty list*.
  Also called `nil` .

  [It is both an atom and a list.]

Properties of Lists

- No need to predefine the *length* of a list.

- Differs from set:
  - can contain multiple instances of the same element
  - elements are in a strict order.

Eg Lists

L1: `(b a c b)`
L2: `(a b b c)`
L3: `(a b c)`

all different

Operations on Lists CONS, CAR, CDR

CONS builds an s-expr from 2 others; CAR, CDR take apart non-empty lists.

- CONS of s_0 and $(s_1 \cdots s_n)$ is
  $(s_0\ s_1 \cdots s_n)$

  2^{nd} arg must be list

  Value always a non-empty list

- CAR of $(s_0\ s_1 \cdots s_n)$ is s_0

  CAR takes a list & returns its first element

- CDR of $(s_0\ s_1 \cdots s_n)$ is $(s_1 \cdots s_n)$

  CDR takes a list & returns all but first element.

  Value is always a list (possible empty).

  [pronounced “could-er”]

Examples of List Operations

cons	of	a	and	(b c)	is
cons	of	a	and	((b c))	is
cons	of	a	and	()	is
cons	of	()	and	()	is
cons	of	a	and	b	is
cons	of	(a b)	and	(c)	is

car cdr	of	(a b c)	is
car cdr	of	(a)	is
car cdr	of	((a b) c)	is
car cdr	of	tom	is
car cdr	of	()	is

N.B. In Franz Lisp:

`car`	of	()	is redefined to be	`nil`
`cdr`	of	()	is redefined to be	`nil`

Predicates

- *Predicate* $\equiv$ A Boolean-Valued Function
  Ie, returns
  $\mathcal{T}$rue or $\mathcal{F}$alse

- In *Lisp*:
  $\mathcal{F}$alse is `NIL`
  $\mathcal{T}$rue is any other s-expr
  (but "T" usually used)

Predicates: `ATOM`, `LISTP`

- `ATOM` tests if s-expr is atomic.
  - `ATOM` of `s` is $\left\{ \begin{array}{ll} \texttt{t} & \text{if } \texttt{s} \text{ is an atom} \\ \texttt{nil} & \text{otherwise} \end{array} \right\}$
  - Eg:

ATOM	of	a	is
ATOM	of	(a)	is
ATOM	of	(a (b) c)	is
ATOM	of	nil	is

- `LISTP` tests if s-expr is a list.
  - `LISTP` of `s` is $\left\{ \begin{array}{ll} \texttt{t} & \text{if } \texttt{s} \text{ is a list} \\ \texttt{nil} & \text{otherwise} \end{array} \right\}$
  - Eg:

LISTP	of	a	is
LISTP	of	(a)	is
LISTP	of	(a (b) c)	is
LISTP	of	nil	is

Predicates: `EQ`, `NULL`

- `EQ` compares two atoms for equality
  - `EQ` of a_1 and a_2 is $\left\{ \begin{array}{ll} \texttt{T} & \text{if same atom} \\ \texttt{NIL} & \text{if different atoms} \end{array} \right\}$
  - Eg:

`EQ`	of	`a`	and	`a`	is
`EQ`	of	`a`	and	`b`	is
`EQ`	of	`a`	and	`a1`	is
`EQ`	of	`a`	and	`nil`	is
`EQ`	of	`nil`	and	`()`	is
`EQ`	of	`a`	and	`(a)`	is
`EQ`	of	`(a)`	and	`(a)`	is

- `NULL` tests if s-expr is empty list
  - `NULL` of `s` is $\left\{ \begin{array}{ll} \texttt{T} & \text{if } \texttt{s} \text{ is } \texttt{nil} \text{ (or } \texttt{()}\text{)} \\ \texttt{NIL} & \text{otherwise} \end{array} \right\}$
  - Eg:

`NULL`	of	`nil`	is
`NULL`	of	`()`	is
`NULL`	of	`a`	is
`NULL`	of	`(a)`	is
`NULL`	of	`( () )`	is

- Note: `NULL` of `nil` is `t` / `NULL` of `t` is `nil` — Like Negation.