## 0.1 Some Introductory Lisp Concepts

Why Lisp?

- oriented toward symbol manipulation
  - automatic facilities for associating information with symbols
  - easily constructed data structures
- more flexible than most programming languages – user's have lots of control over how they choose to program, can tailor things to look the way they want.

Where Different

1. Lisp is an interpreted language – so can get immediate feedback. The functions you write can be compiled, but usually this is not done until all debugging is finished.
2. Everything is viewed as a list. Programs and data look the same.
3. Language gives natural functions for dealing with lists – provides list manipulation functions to put lists together, take them apart, walk through lists, etc...
4. As a programmer, you are encouraged to program in a recursive style.

The Data Structures – The basic data structure is termed an s-expression (symbolic expression). They come in two (three) flavors:

1. atoms – numbers and identifiers (where identifiers is a combination of numbers and letters and some other special symbols)
2. lists = ( s-expression1 s-expression2 ... s-expressionn) n $\geq$ 0, empty list = (), nil.
3. dotted pair = ( s-expression . non-nil-atom)

s-expressions
a
a2635
(A)
()
((A))
(a b)
((A) (B))
(((a) b))
a.b (no good!)
(a . b . c) (no good!).

<u>Evaluation Rules</u>

1. Atom
   (a) number – the value of a number is that number itself.
   (b) t or nil – evaluate to themselves (t and nil are special atoms).
   (c) any other identifier – look up value of it as a variable.

   [So, at this point we know we can type numbers and t or () or nil at lisp and it will return the same thing back. We can't yet do identifiers since we don't know how to set their values yet.]

2. Lists – The first element of a list denotes a function (or operator) name and the remaining elements are its arguments. There are two cases:
   (a) Normal Function – evaluates all arguments in a left to right order, and then applies the function to the resulting values.
   (b) Special Functions – use special rules to evaluate the arguments, the function is then applied to the arguments or values.

   We will call the object present to a call to a function the *supplied arguments.* We call the values upon which the computation is performed, the *actual arguments.*