Adrian Velonis, Haley Nolan, and Maya Johnson
Principles of Programming Languages
Deepak Kumar
October 26, 2020

Assignment #3

Structure of a Lisp Program

All Lisp programs are made up of S-expressions, which in turn are made up of lists or atoms (everything that's not a list).

Types and Operators

- Integer Types: bignum - true integer type, only limited by memory; fixnum - implementation-dependent, minimum range is -2^{15} to $2^{15}-1$ (i.e., ≥ 16 bits)
- Rational Type: ratio - uses 2 bignum values, 1 numerator and 1 denominator
- Floating Point Types: Precision is implementation-dependent; short-float ≥ 13 bits, single-float ≥ 24 bits, double-float ≥ 50 bits, long-float ≥ 50 bits
- Boolean Type: Value nil is false; all other values are true (usually t)
- Logical Operators: and, or, not
- Character Type: #\x represents character 'x'; no numeric comparison, but can be compared to other characters with CHAR=, CHAR/=, CHAR<, CHAR>, etc.
- String Type: A one-dimensional array of characters; Enclosed in "" (e.g. "Bryn Mawr"); Can be compared using STRING=, STRING/=, STRING<, STIRNG>, etc
- Comparison Operators (for non-characters or strings): =, /=, >, <, >=, <=, eql (checks type)

Variables and Declaration

Variable names:

Common Lisp variable names are made of lower case letters and can include numbers, with words being seperated by "-" (e.g. bryn-mawr). Global ("special") variables are conventionally marked using "*" (e.g. *bryn-mawr*).

Scope:

Local variables are lexically scoped, and dynamic variables (Common Lisp version of global variables) are dynamically scoped.

There is no type declaration in Lisp.

Control Structures

Assignment:

Local variables -
Defined with “let”

```
(let (<var> <expression>))
```

Ex: `(let ((str "Hello, world!"))`

Local variables can then be assigned new values using “setq” or “setf.” Preference and convention for which one is used varies.

```
(setq <existing var> <new value/expression>)
(setf <existing var> <new value/expression>)
```

Can use “lambda” as a shorthand declaration for lambda expressions

```
(<function call> (lambda <var> <expression>))
```

Dynamic variables -
Defined with “defvar” or “defparameter”

Defparameter requires an initial value, and defparameter variables change if code with a new initial value is reloaded

```
(defvar <var> <expression>)
(defparameter <var> <expression>)
```

You can redefine dynamic variables using the let keyword - the new value will be visible only within the let function.

Multiple Assignment:

Multiple variables using “let” -

```
(let ((<var1> <expression1>)
     (<var2> <expression2>))
```

Multiple variables where one depends on previous variables -

```
(let* ((<var1> <expression1>)
     (<var2> (operation including var1)))
```

Multiple variables using “multiple-variable-bind” -

```
(multiple-value-bind <var-1 .. var-n> <expression>
     <body that uses the variables (optional)>)
```

Conditional Statements - if:

All of the following function as conditional statements in Common Lisp.

```
(if <condition> <value if true> <value if false>)
(cond (test then) (t else))
(when <condition> <value>)
(unless <condition> <value>)
```

Ex:

```
(if t 5 6)
  5
(cond ((= 5 6) 5) (t 6))
  6
(when t 5)
  5
(unless t 5)
  NIL
```

Loops:

Iteration can be done in Common Lisp through some variation of a `for` loop, or through `iter`.

```
(loop for loop-variable in <a list>
   do (action))
```

Example: `(loop for x in '(1 2 3))`

```
(loop for loop-variable from value1 to value2
   do (action))
```

Example: `(loop for x from 1 to 100)`

Common Lisp lacks a `while` loop, but this functionality can be replicated by defining a macro of that name.

```
(defmacro while (condition &body body)
   (loop while ,condition do (progn ,@body)))
```

Usage:

```
(while (some-condition)
  (do-something)
  (do-something-else))
```

Iter:
The “iter” construct can also be used to iterate through multiple items in a set. It must be imported from the “iterate” library.

```
(iter (for i from 1 to 5))
```

Recursion

Example of recursion in CL:

```
(def factorial (x)
     (cond (= x 1 1)
     (t (* x (factorial (- x 1))))))
```

Resources

- https://lisp-lang.org/learn/getting-started/
- https://lispcookbook.github.io/cl-cookbook/iteration.html
- https://lispcookbook.github.io/cl-cookbook/numbers.html
- https://www.tutorialspoint.com/lisp/index.htm
- https://www.cs.cmu.edu/~15110-f12/Touretzky-Common-Lisp-ch8.pdf
- https://en.wikibooks.org/wiki/Common_Lisp
- http://www.lispworks.com/documentation/HyperSpec/Body/t_ban.htm
- https://www.cs.cmu.edu/Groups/AI/html/cltl/clm/node75.html
- http://clqr.boundp.org/clqr-a4-consec.pdf
- http://www.gigamonkeys.com/book/
- http://n-a-n-o.com/lisp/cmucl-tutorials/LISP-tutorial-17.html
- https://stackoverflow.com/questions/3855862/setq-and-defvar-in-lisp
- http://www.lispworks.com/documentation/HyperSpec/Body/m_lambda.htm
- http://www.lispworks.com/documentation/HyperSpec/Body/m_multip.htm#multiple-value-bind