

Basics of Using Lisp

Getting into and out of Clisp

- ◇ Entering **% clisp**
- ◇ Do Lisp work
- ◇ Exiting **(bye)**
 - » **A function with no arguments**
 - » **CTRL-d** can also be used
- ◇ Documentation files are in the directory
 - » **/cs/local/doc/clisp**

Do Lisp work

- ◇ Edit in files with extension “.lsp”

- ◇ Load the files

 - (**load 'filename.lsp**)

 - > **Loading executes the S-expressions in the file**

 - > **Loading defines symbols**

 - > **No other computational effects are seen**

 - > **Comments begin with “ ;”**

- ◇ Interactively execute S-expressions

 - » **Invoke functions**

 - » **Use setq to define symbol values**

 - » **Use defun to define functions**

The Lisp Interpreter

- ◇ Is a loop over the following functions
 - » **Read**
 - > **an S-expression**
 - » **Eval**
 - > **the S-expression**
 - » **Print**
 - > **the result**
- ◇ You can redefine these functions !!!
 - » **But it is dangerous if you do not know what you are doing.**

Free and bound variables


◇ Free variable

- > global variable defined at the upper level
(outside of function definitions)
- » Within a function consistently changing the name of a free variable will normally alter the semantics of the function

`(defun x (a) (+ a y))`

is NOT the same as

`(defun x (a) (+ a z))`



Bound variables

◇ Bound variable

- > **local variable defined in the parameter list of a function**
- » **Within a function consistently changing the name of a bound variable will not alter the semantics of the function**

is the same as

```
(defun x (a y) (+ a y))  
  
(defun x (a z) (+ a z))
```

Static & Dynamic scoping

- ◇ Consider the following

```
( defun f1 ( v1 ) ( f2 v1 ) ) ;; v1 defined – argument  
( defun f2 ( v2 ) ( 1+ v1 ) ) ;; is v1 defined?  
( f1 7 )
```

- ◇ Under **static (lexical) scoping** invoking (f1 7) produces an error as **v1** is undefined in **f2**
- ◇ Under **dynamic scoping** **v1** in **f2** is defined because **f2** is executed in the environment of **f1** in which **v1** is defined
 - » **Dynamic scoping leads to the funarg problem as function arguments can shadow (hide) global variables**

Execution Environment

- ◇ An environment consists of binding between a set of **symbols** and their **values**

((**A** **1**) (**B** **5**) ... (**D** (**a** **b** **c**)))

- ◇ At the interpreter level global symbols are created, **using setq or defun**, giving a global environment

- ◇ The value of a symbol is looked up in the environment

- ◇ Evaluating a function causes the parameters to be **prepended** to the **appropriate** environment

» **Evaluating** (f1 3) **defined as** (defun f1 (v1) (f2 v1))

» **creates the environment**

((**v1** **3**) (**A** **1**) (**B** **5**) ... (**D** (**a** **b** **c**)))

Execution Environment – 2

- ◇ We evaluate **(f2 v1)** in the context

((v1 3) (A 1) (B 5) ... (D (a b c)))

- ◇ **v1 has the value 3 – passed as an argument to f2**

- ◇ **f2** is defined as

(defun f2 (v2) (1+ v1))

- ◇ What environment does f2 use?

» **We have two choices**

> **Dynamic scoping**

> **Static scoping**

Execution Environment – 3

◇ Dynamic scoping

» **passes the existing environment**

((v1 3) (A 1) (B 5) ... (D (a b c)))

» **after prepending (v2 3)**

– v2 is the parameter of f2 and 3 is the argument from f1

> **The following is passed**

((v2 3) (v1 3) (A 1) (B 5) ... (D (a b c)))

> **So v1 has a definition in the environment**

» **The environment grows and shrinks on entry and exit from functions**

> **A different environment for every function**

Execution Environment – 4

◇ Static scoping

- » **passes the environment in the context of the definition of f2**
- » **the same environment passed to f1**
- » **after prepending (v2 3)**
 - > **The following environment is passed**
((v2 3) (A 1) (B 5) ... (D (a b c)))
 - > **So v1 has NO definition in the environment**
- » **Environment on entry is fixed by the static structure**
 - > **The same environment for every function**

Dynamic scoping funarg problem

- ◇ Dynamic scoping leads to the **funarg** problem as function arguments can shadow global variables
 - » (**defun funarg** (**func arg**) (**funcall func arg**))
 - » (**defun timesArg** (x) (* arg x))
 - » (**setq arg** 2)
 - » (**funarg** 'timesArg 3)
- ◇ In a static environment the result is 6
- ◇ In a dynamic environment the result is 9
- ◇ Is Clisp dynamically or statically scoped?

Do Lisp Work ... **Reminder**

**Do not use `setq` within
function definitions**

Setq creates global symbols NOT local symbols

Very poor programming practice