Selection in lisp

- Lisp provides many functions to select actions based on true/false conditions
- *if* is used for if/else functionality
- *cond* is used for chains of if/else if/else if/.../else
- *case* is used much like a C switch statement
- *typecase* is like case, but based on an item’s data type
- *when* allows you to do multiple things if a condition is true
- *unless* allows you to do multiple things if a condition is false

If for selection in lisp

- The if function syntax is (if condition trueval falseval)
- If the condition evaluates to true then the trueval is returned, otherwise the falseval is returned
- The falseval defaults to nil if omitted
- Any value or function call is valid for the trueval/falseval
- Example: if x is a number return its square root, otherwise return the string “not a number”:
- (if (numberp x) (sqrt x) “not a number”)

Nested if's

- Suppose we wanted functionality like:

```
if x is a number
   if x < 0 return sqrt(-x)
   else return sqrt(x)
else if x is a string return length(x)
     else return nil
```

- One solution:

```
(if (numberp x)
    (if (< 0 x) (sqrt (- x)) (sqrt x))
    (if (string x) (length x) nil))
```

Sequences of if/else-ifs

- Suppose we want functionality like

```
If (a) w
Else if (b) x
Else if (c) y
Else z
```

- Could use nested if’s:

```
(if a w
      (if b x
            (if c y z)))
```

Cond: alternative to nested if’s

- Cond is meant as an alternative to the nested-if syntax
- You list a series of pairs, for each pair there is a condition and then the value to return if the condition is true
- The cond returns the result of the first match

```
(cond
   (a w)
   (b x)
   (c y)
   (t z))
```

- Note the t as the final condition acts like a final else

Cond example

```
; sample cond layout, note the bracketting
(cond
   ((not (numberp x)) x) ; if x isn't a number return x
   ((< 0 x) (* x 10))    ; else if x<0 return 10* x
   (t (x - 5)))          ; else return x-5
```

Compound expressions

- Boolean expressions and, or, not supported, e.g.
- (and x y z)
- (or a b c d e f)
- (not x)
- (and (not (or a b c)) (or x y z))

When blocks

- When allows you to test a condition and do multiple things if it is true, when’s return value is the last return value in the block

```
(when (< x 0)
   (format t “~A negative, replacing with abs value~%”)
   (setf x (- x)))
```

Unless blocks

- Unless allows you to perform multiple actions if a condition is false

```
(unless (< x 0)
    (format t “setting y to root x~”)
    (setf y (sqrt x)))
```

Case statements

- Act much like a switch in C/C++

```
(case x
    (0 (format t “x is 0”))
    (“foo” (format t “x is foo”))
    (otherwise (format t “x is something else”)))
```

- Basically like a cond where each test condition is “does x equal this?”

Typecase statements

- Like case, but works on type of item instead of its value

```
(typecase x
    (string (format t “x is a string”))
    (number (format t “x is a number))
    (t (format t “x is something else”)))
```