Welcome!

- Programming in Lisp
  - Course Number CSCI 2210-01
  - Class Meetings
    - Sage 3303
    - Aug. 24 - Oct. 14
    - Wednesdays, 4 - 6 pm

Instructor

- Kenneth W. Flynn
  - BS Physics
  - MS Computer Science (May '99)
  - Ph. D Astrophysics (Someday)
  - Email: flynnk@rpi.edu
  - Office Hours

Texts

- *Ansi Common Lisp* by Paul Graham; Prentice Hall, 1996
  - Required
- *Lisp* by Guy L. Steele
  - Optional

Grading

- Two schemes...

<table>
<thead>
<tr>
<th>Item</th>
<th>Weights</th>
<th>Alternative Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>Halfway Quiz</td>
<td>15%</td>
<td>15%</td>
</tr>
<tr>
<td>Final Exam</td>
<td>15%</td>
<td>-</td>
</tr>
<tr>
<td>Class Participation</td>
<td>25%</td>
<td>25%</td>
</tr>
<tr>
<td>Homework #1</td>
<td>15%</td>
<td>20%</td>
</tr>
<tr>
<td>Homework #2</td>
<td>15%</td>
<td>20%</td>
</tr>
<tr>
<td>Homework #3</td>
<td>15%</td>
<td>20%</td>
</tr>
</tbody>
</table>

Course Notes

- Course notes and other materials will be available via the course website
  - [http://www.cs.rpi.edu/courses/fall98/lisp/](http://www.cs.rpi.edu/courses/fall98/lisp/)
- Check the website frequently

Kenneth Flynn (1-6)
09/08/98
Course Policies (I)

- Homework
  - Due at 11:59:59 on date given in syllabus and on assignment
  - Late homework is penalized 10% for each RPI class day late. Extensions may be requested until noon of the date due
  - Students may work in teams of 2. Both students will receive the same grade

Course Policies (II)

- Exams
  - May cover material from lectures or readings
  - Time conflicts should see me ASAP

And now...

- Lisp!
- Name comes from LIST Processor
- Lisp is... well, different
- In Lisp, data == programs
- So why learn Lisp?
  - AI uses
  - Different (buzzword alert!) paradigm for programming

Lisp

- Lisp is interactive
- Most of the time when Lisp is waiting for input, we say it is at the toplevel.
- We can type lisp expressions into the toplevel and they will be evaluated:

\[
\begin{align*}
&> \ 1 \\
&\text{1}
\end{align*}
\]

Atoms & Lists

- Atoms
  - A single element of a particular data type
- Lists
  - Lists may contain atoms or other lists
  - Enclosed in parentheses "(atom atom atom list)"
  - We'll talk more about lists later, and a lot more next week

Atoms & Lists Examples

- Atoms
  - 1
  - 3.3
  - WUMPUS
- Lists (LISP)
  - (1 2 3)
  - (1 3 FIVE)
  - (A LIST (A NESTED LIST))
Expressions

- Syntactic structure
- (Operator Argument Argument ...)
- That's 0 or more arguments
- Expressions are lists! (Ponder this a moment!)

Expression Examples

- (> (*))
  1
- (> (* 1))
  1
- (> (* 2))
  2
- (> (* 2 2))
  4
- (> (* 1 2 3))
  6

Evaluation

- Evaluation (what happens when you press enter) happens in two steps:
  - The arguments are evaluated, left to right
  - Call by value occurs for the given operator (function)
- This is the Evaluation Rule for Common Lisp

Evaluation Example

- (> (/ (* 4 6) 3))
  - / is the operator, skip this
  - (*) 4 6 is the first argument, let's evaluate
  - * is the operator, skip this
  - 4 is the first argument, it evaluates to itself
  - 6 is the second argument, it evaluates to itself
  - * is now called with the arguments 4 and 6
  - (* 4 6) is replaced by 24
  - We currently have: (/ 24 8)
  - The second argument to / is 8, it evaluates to itself
  - / is now called with the arguments 24 and 8
  - (/ 24 8) is replaced by 3
  - This is returned to the toplevel

Evaluation Example II

- We currently have (/ 24 8)
- The second argument to / is 8, it evaluates to itself
- / is now called with the arguments 24 and 8
- (/ 24 8) is replaced by 3
- This is returned to the toplevel

More Expression Examples

- (> (/ (* 4 6) 3))
  8
- > (quote hello)
  HELLO
- > hello
  HELLO
**The Quote Operator**
- These are equivalent:
  - `(quote Hello)`
  - `'Hello`
- Special Operator
  - Disobeys the Evaluation Rule
  - Quote says "Don't evaluate my argument"

**Quote Examples**
```
> (quote hello)
HELLO
> 'hello
HELLO
> hello
;: Error: Unbound variable HELLO in #<function 1 #x810FC0>
```

**Symbols**
- When Lisp returns something like:
  - `> 'ARTICHOKE`
  - `ARTICHOKE`
  - This is a symbol
- We'll talk more about symbols later, but for now...
- Symbols are names for other things. One role they fill is that of variables.

**The Story So Far...**
- Atoms
- Lists
- Expressions
- Evaluation (The Evaluation Rule)
- Symbols
- `'

**list**
- Another operator
- This one builds lists
- `> (list 1 2 3)`
  - `(1 2 3)`
- `> (list 1 (+ 1 1) 3)`
  - `(1 2 3)`
- `> (list 'Tada (+ 1 1) 3)`
  - `(TADA 2 3)`

**list II**
- These do the same thing:
  - `(list 1 2 3)`
  - `(1 2 3)`
nil

- Symbol
- Represents empty list -- a list with no elements
- > 'nil
  NIL
- > nil
  NIL
  - nil evaluates to itself
- '()  NIL

Is There No Truth in Beauty?

- Lisp has the concept of Boolean values
- The value "True" is represented by t
  - > t
    T
- The value "False" is represented by nil
  - This is a second use for nil
- Functions that determine truth are called predicates

Predicates: listp

- listp
  - Is the argument a list?
  - > (listp 'Beauty)
    NIL
  - > (listp '(No Lie))
    T

Predicates: null

- null
  - Is the argument an empty list?
  - > (null nil)
    T
    - > (null '(The Truth Is Out There))
      NIL

Predicates: not

- not
  - Returns opposite of the argument
  - > (not t)
    NIL
  - > (not nil)
    T
  - Does exactly the same as null. For readability, though...

"if", "and", "or", but no "but"s

- These statements begin to allow for logic in your programs
- if is the simplest form of flow control (which is somewhat different than in iterative languages)
**If**

- if 'Truth-Statement 'Then-do-this 'Else-do-this
  - if is a macro. Macros disobey the Evaluation Rule
  - (The exceptions prove the rule?)
  - For if: The first argument is always evaluated
  - If the first argument is true, the second argument is evaluated, and its value returned
  - If the first argument is false, the third argument is evaluated instead, and its value is returned.

**if examples**

- >(if t 'A 'B)
  - A
- >(if nil 'a 'b)
  - B

**and / or**

- and
  - Macro
  - Returns true if all arguments are true
  - Lazy evaluation (stops at first false argument)
- or
  - Macro
  - Returns true if any argument true
  - Lazy evaluation (stops at first true argument)

**Functions (defun)**

- Create new functions with defun.
  - Syntax: (defun function-name (parameter-list) (expressions))
  - Macro
  - Functions make up the majority of functionality provided

**Function Examples**

- >(defun adder (x y)
  - (+ x y))
- >(adder 3 2)
  - 5
- >(adder 1.0 3.5)
  - 4.5

**Review**

- Atoms, Lists, Expressions, Evaluation (The Evaluation Rule), Symbols, '
- Predicates (listp, null, not, and, or)
- Macros (conceptually)
- Functions (practically)
- This class was an introduction of a lot of concepts. From now on, we'll be more focused.
On the Next Exciting Episode!
- Input and Output
- Variables (or not...)
- Lisp Data Structures: lists and arrays

For Next Week...
- Read Chapter #2 in Graham
  - We didn't cover everything in Chapter #2
    - Some we will do next week
    - Some we will come back to

Operations on Lists
- Lots of this next week, but for now:
  - car returns the first element
  - cdr returns a list containing everything except the first element

```lisp
> (car '(1 2 3))
1
> (cdr '(1 2 3))
(2 3)
```

Operations on Lists II
- cons builds a list. It takes the first argument and attaches it to the beginning of the second argument:

```lisp
> (cons 1 '(2 3))
(1 2 3)
> (first '(1 2 3))
1
> (second '(1 2 3))
2
```