

Structure of Programming Languages – Lecture 9

CS 636 – 536

June 8, 2009

- 1 Mapping Functions
 - Concepts
 - Maps in the Old Languages
 - Maps in Languages

Definitions

In their simplest form, **map functions** ...

- Are higher-order functions
- Take a function and a list or array of data as parameters
- Apply the function to each data element.
- Produce either a number or a list as a result.

More complex map functions ...

- Take more than one functional parameter.
- Or take more than one array or list parameter.

Ways to Map

These mapping patterns (and many variations) are from LISP:

- `(mapcar <function> <sequence>)`
Apply the unary function to each element of the sequence.
Return a list of the results.
- `(mapcar <function> <sequence1> <sequence2>)`
Apply the binary function to corresponding pairs of elements from the two sequences. Return a list of the results.
- `(maplist <function> <sequence>)`
Apply the function to the whole sequence, then to successive `cdr`'s of the sequence. The result is a list.
- `(reduce <function> <sequence>)`
Apply the binary function to the first pair of elements from the sequence, then to the result and the next sequence element, etc. Return the result from the final application of the function.

LISP examples

Assume we have the following sequences:

- L1 = (1 2 3 4)
- L2 = ()

Expression	Result
<code>(mapcar '*2 L1)</code>	<code>(2 4 6 8)</code>
<code>(mapcar '*2 L2)</code>	<code>()</code>
<code>(mapcar '* L1 '(2 3 5 7))</code>	<code>(2 6 15 28)</code>
<code>(maplist '+ L1)</code>	<code>10 9 7 4</code>
<code>(reduce '* L1)</code>	<code>24</code>

APL examples

$$A \leftarrow 0\ 1\ 0\ 1 \quad B \leftarrow 3\ 5\ 7\ 11 \quad C \leftarrow 1\ 2\ 3$$

$$M \leftarrow 3\ 2\ \rho\ \iota\ 6 \quad N \leftarrow 2\ 2\ \rho\ 0\ 1\ 1\ 2$$

Expression	Result	Name and Action
$+/\ A$	2	Reduce: like LISP, add all elements.
$\wedge/\ A$	0	Test for all elements = true.
$+\ \backslash\ A$	0 1 1 2	Scan: apply operator to the 1st element, then to the 1st and 2nd, etc, until finally applying it to all elements in the array. Collect the results in an array.
$\vee\ \backslash\ A$	0 1 1 1	
$C\ \circ.\ +\ B$	4 6 8 12 5 7 9 13 6 8 10 14	Outer "product" of two vectors. Result is a matrix where left operand controls # of rows, right controls # of columns.
$M\ +.\ \times\ N$	2 5 4 11 6 17	Inner product or Matrix product. Columns in M must match rows in N.

Maps Around the World

Erlang	<pre>map (X, Y) <- map(X, Y, _). map (_, [], Z) <- Z; map (X, Y, Z) <- map(X, tl(Y), list:append(Z, X(hd(Y)))).</pre>
Smalltalk	<pre> #(1 2 3 4 5) collect: [:each 2 * each]</pre>
Haskell	<pre>map (2*) [1,2,3,4,5]</pre>
Ruby	<pre>[1, 2, 3, 4, 5].map x 2*x</pre>
Python	<pre>sequence = [1,2, 3, 4] answer = [] def func(x): return x*x for(k in sequence) { answer.append(func(k)) }</pre>
Python	<pre>map(func, sequence)</pre>
Java	<pre>for(int k : sequence) {...}</pre>

List Comprehensions

A **list comprehension** is an expression that denotes a list. Miranda and Python both support list comprehensions that can be easily translated into code.

- `answer = [x*x for x in sequence]`
The same list as the for loop in the previous slide.
- `squares = [x**2 for x in range(10)]`
The first ten square numbers: [0, 1, 4, 9, ... 64, 81]
- `evens = [x for x in range(10) if x%2 == 0]`
The first five even numbers: [0, 2, 4, 6, 8]
- `wacky= [x+y for y in [10, 20, 30] for x in range(8) if x%2 == 0]`
The y loop is the outer loop and the x loop is the inner loop.
Result: [10, 12, 14, 16, 20, 22, 24, 26, 30, 32, 34, 36]