STLRUBY
meet[:up]

Conses in Ruby: So Much More Than Lists

Christopher Mark Gore

cgore.com

Monday, September 12, AD 2016

Ruby is my ~~most second~~ third favorite programming language of all time.

1. My own super-awesome programming language, Teepee *(but it's not that awesome just yet)*
2. Common Lisp
3. Ruby

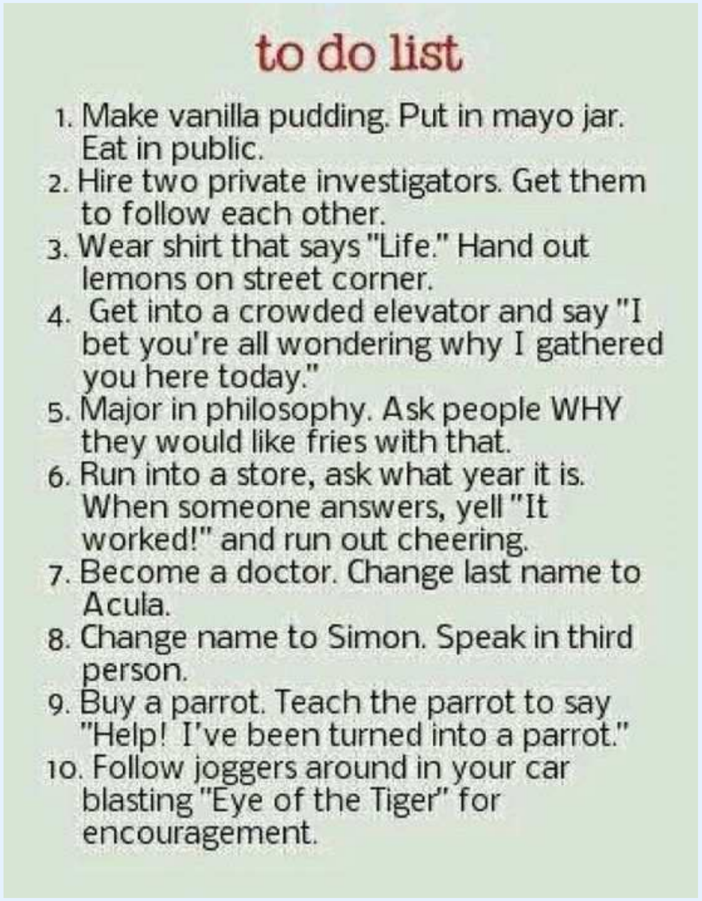
.

.

.

999. Java

Nearly every programming language has some way to do things to a list/array/vector/whatever of things.



to do list

1. Make vanilla pudding. Put in mayo jar. Eat in public.
2. Hire two private investigators. Get them to follow each other.
3. Wear shirt that says "Life." Hand out lemons on street corner.
4. Get into a crowded elevator and say "I bet you're all wondering why I gathered you here today."
5. Major in philosophy. Ask people WHY they would like fries with that.
6. Run into a store, ask what year it is. When someone answers, yell "It worked!" and run out cheering.
7. Become a doctor. Change last name to Acula.
8. Change name to Simon. Speak in third person.
9. Buy a parrot. Teach the parrot to say "Help! I've been turned into a parrot."
10. Follow joggers around in your car blasting "Eye of the Tiger" for encouragement.

Ruby has arrays.

```ruby
a = [1,2,3,4,5]
a.class # Array
a.length # 5
a.first # 1
a.map {|i| i*2} # [2,4,6,8,10]
```

Common Lisp prefers linked lists.

```
(setf a '(1 2 3 4 5))
(class-of a) ; #<BUILT-IN-CLASS COMMON-LISP:CONS>
(length a) ; 5
(first a) ; 1
(mapcar (lambda (i) (* 2 i)) a) ; '(2 4 6 8 10)
```

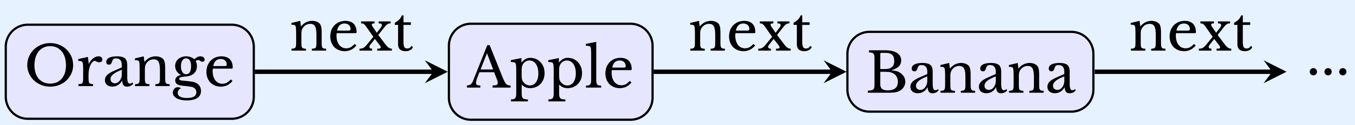
But Common Lisp also has vectors, which are basically the same as Ruby arrays.

```lisp
(let v (vector 1 2 3 4 5))
(class-of v)
;; #<BUILT-IN-CLASS COMMON-LISP:SIMPLE-VECTOR>
(length v) ; 5
(elt v 0) ; 1
(map 'vector (lambda (i) (* 2 i)) v)
;; #(2 4 6 8 10)
```

And I’m adding a really nice Cons gem!

```
require 'cons'
c = [1,2,3,4,5].to_cons
c.class # Cons
c.length # 5
c.first # 1
c.map {|i| i*2} # Cons.list 2,4,6,8,10
```

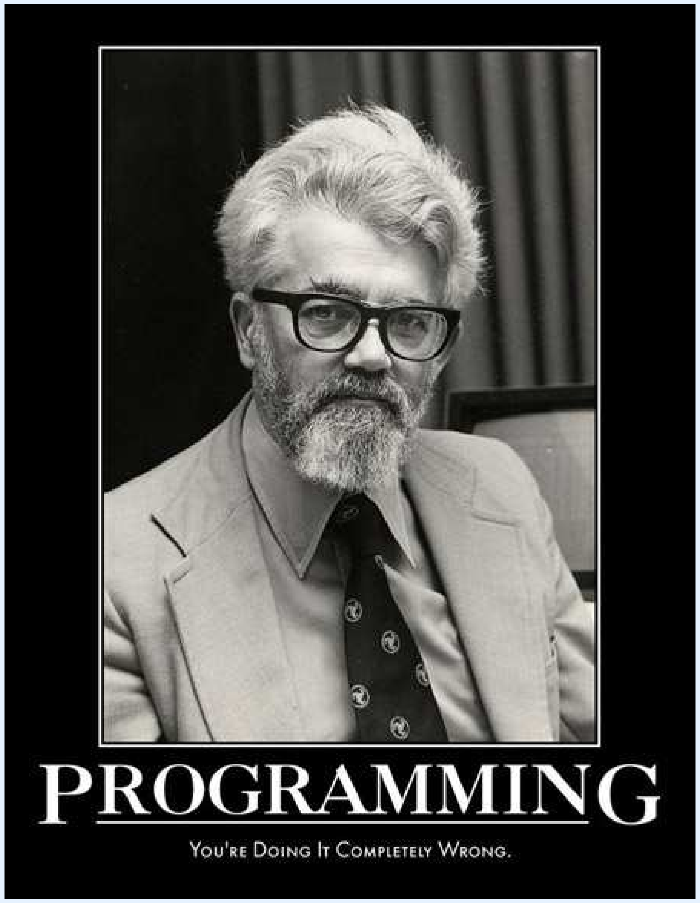
Remember linked lists?



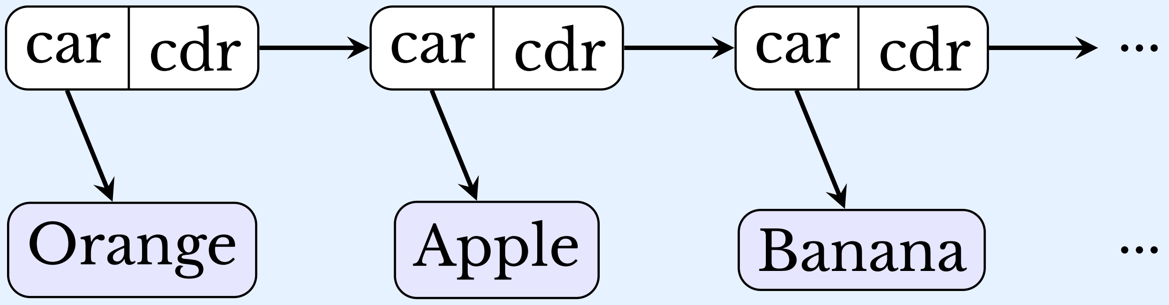
We can do this something like this in Ruby:

```
class LinkedList
  attr_accessor :data, :next
  def initialize data, next
    @data, @next = data, next
    ...
  end
  ...
end
```

Well, they don’t work like that at all in Lisp.



Lisp uses conses instead.
The data isn't in the cell, it's two pointers.

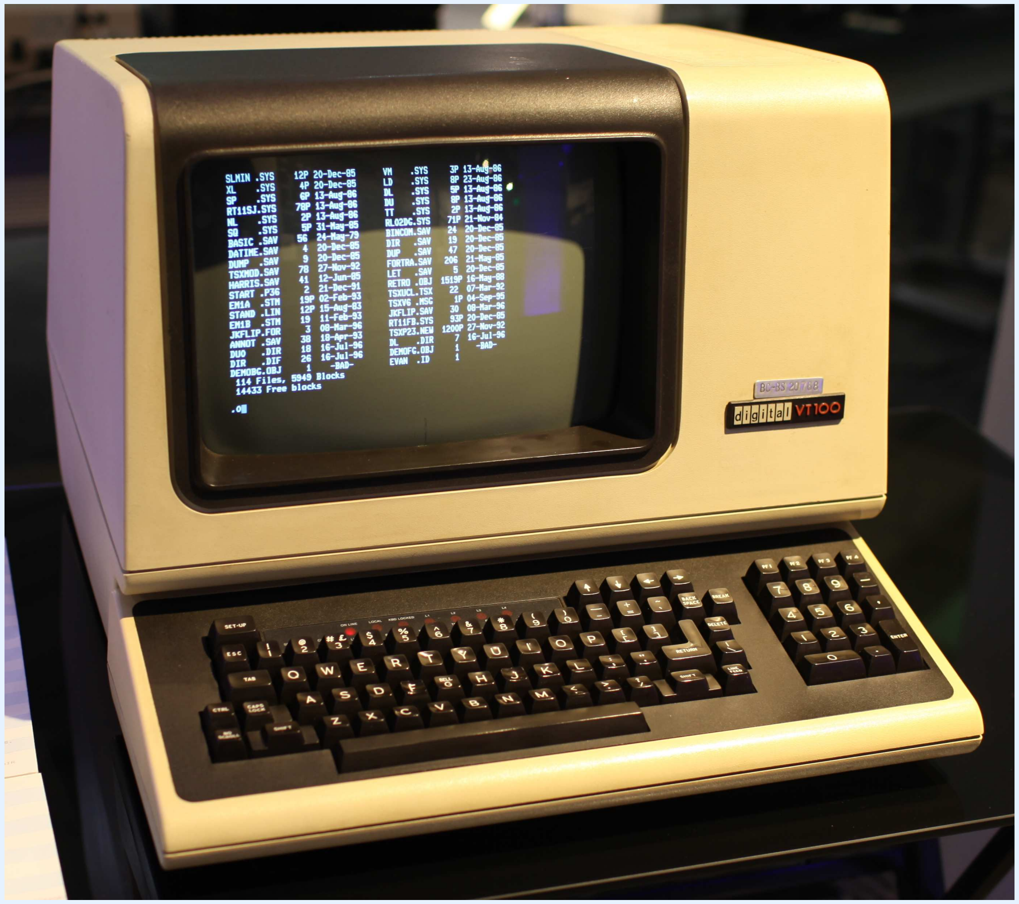


Say hello to the IBM Type 704

`CAR`: Contents of the Address Register

`CDR`: Contents of the Data Register

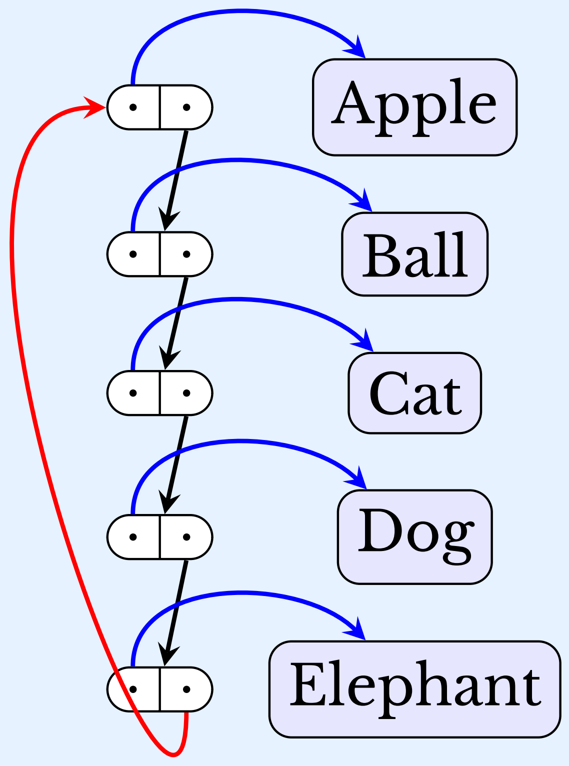
For similar reasons, every Linux box thinks your on one of these:



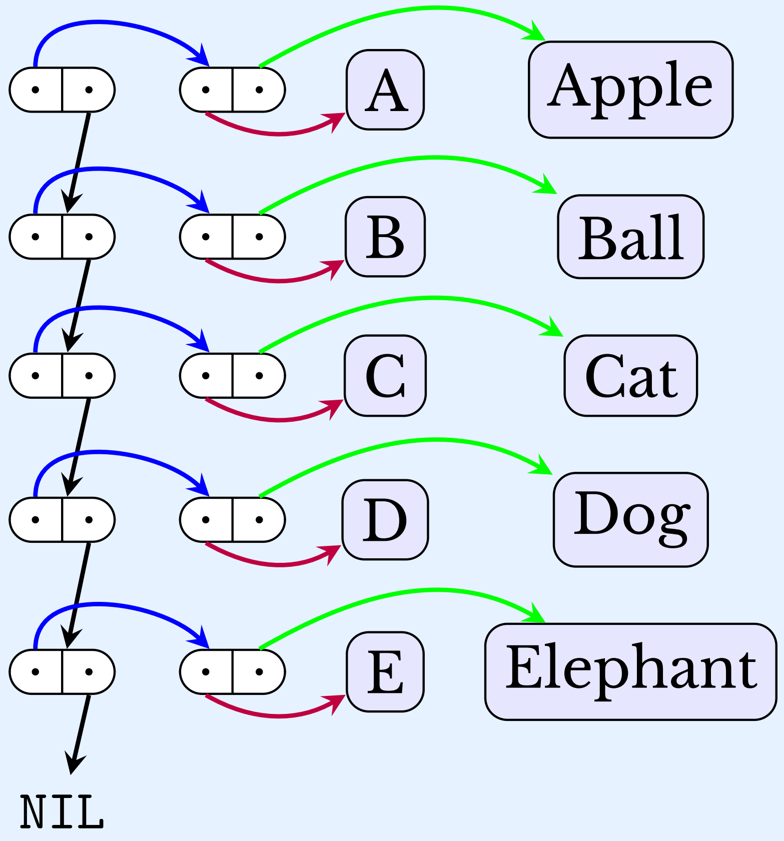
```
SLMIN .SYS    12P 20-Dec-85    VM    .SYS      3P 13-Aug-86
XL    .SYS     4P 20-Dec-85    LD    .SYS      8P 23-Aug-86
SP    .SYS     6P 13-Aug-86    DL    .SYS      5P 13-Aug-86
RT11SJ.SYS    78P 13-Aug-86    DU    .SYS      8P 13-Aug-86
NL    .SYS     2P 13-Aug-86    TT    .SYS      2P 13-Aug-86
SQ    .SYS     5P 31-May-85    RL02DG.SYS     71P 21-Nov-84
BASIC .SAV    56  24-May-79    BINCOM.SAV     24  20-Dec-85
DATIME.SAV     4  20-Dec-85    DIR   .SAV     19  20-Dec-85
DUMP  .SAV     9  20-Dec-85    DUP   .SAV     47  20-Dec-85
TSXMOD.SAV    78  27-Nov-92    FORTRA.SAV    206  21-May-85
HARRIS.SAV    41  12-Jun-85    LET   .SAV      5  20-Dec-85
START .P36     2  21-Dec-91    RETRO .OBJ  1519P 16-May-88
EM1A  .STM    19P 02-Feb-93    TSXUCL.TSX     22  07-Mar-92
STAND .LIN    12P 15-Aug-83    TSXV6 .MSC      1P 04-Sep-95
EM1B  .STM    19  11-Feb-93    JKFLIP.SAV     30  08-Mar-96
JKFLIP.FOR     3  08-Mar-96    RT11FB.SYS     93P 20-Dec-85
ANNOT .SAV    38  18-Apr-93    TSXP23.NEW   1200P 27-Nov-92
DUO   .DIR    18  16-Jul-96    DL    .DIR      7  16-Jul-96
DIR   .DIF    26  16-Jul-96    DEMOFG.OBJ      1    -BAD-
DEMOBG.OBJ     1    -BAD-      EVAN  .ID       1
 114 Files, 5949 Blocks
 14433 Free blocks

.o
```

Why two pointers?
Because you can make ring buffers.



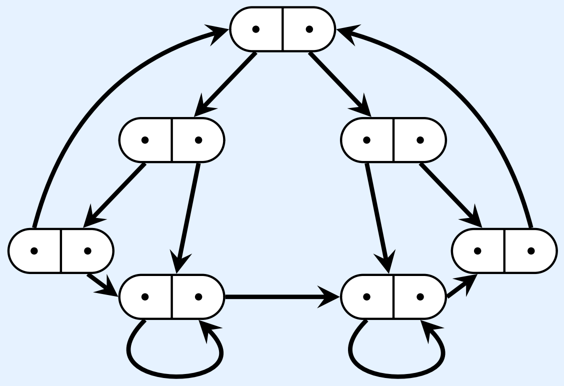
Why two pointers?
Because you can make alists.



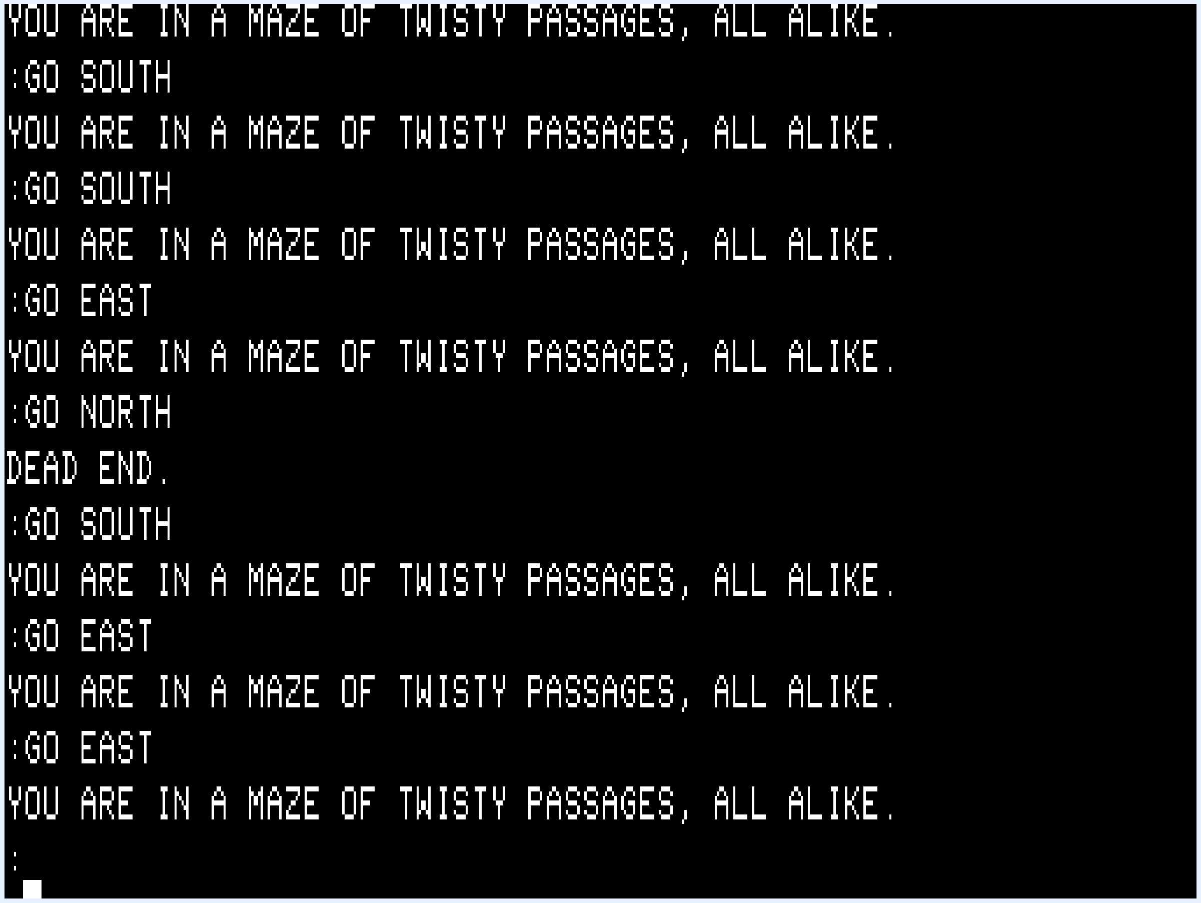
Alists, *association lists*, are how Lisp historically would create things we'd typically use hash maps for today.

```
'((A . Apple)
  (B . Ball)
  (C . Cat)
  (D . Dog)
  (E . Elephant))
```

Why two pointers?
Because then you can do this:



Sometimes, exploring your data structures should be an adventure!



```
YOU ARE IN A MAZE OF TWISTY PASSAGES, ALL ALIKE.
:GO SOUTH
YOU ARE IN A MAZE OF TWISTY PASSAGES, ALL ALIKE.
:GO SOUTH
YOU ARE IN A MAZE OF TWISTY PASSAGES, ALL ALIKE.
:GO EAST
YOU ARE IN A MAZE OF TWISTY PASSAGES, ALL ALIKE.
:GO NORTH
DEAD END.
:GO SOUTH
YOU ARE IN A MAZE OF TWISTY PASSAGES, ALL ALIKE.
:GO EAST
YOU ARE IN A MAZE OF TWISTY PASSAGES, ALL ALIKE.
:GO EAST
YOU ARE IN A MAZE OF TWISTY PASSAGES, ALL ALIKE.
:
```

We can easily do `Cons` in Ruby

```ruby
class Cons
  attr_accessor :car, :cdr
  def initialize car, cdr
    @car, @cdr = car, cdr
    ...
  end
  ...
end
```

Questions?