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.

```
1 (setf a '(1 2 3 4 5))
2 (class-of a) ; #<BUILT-IN-CLASS COMMON-LISP:CONS>
3 (length a) ; 5
4 (first a) ; 1
5 (mapcar (lambda (i) (* 2 i)) a) ; '(2 4 6 8 10)
```
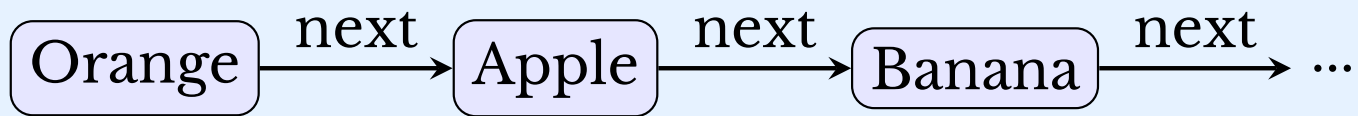
But Common Lisp also has vectors, which are basically the same as Ruby arrays.

```
1 (let v (vector 1 2 3 4 5))
2 (class-of v)
3 ;; #<BUILT-IN-CLASS COMMON-LISP:SIMPLE-VECTOR>
4 (length v) ; 5
5 (elt v 0) ; 1
6 (map 'vector (lambda (i) (* 2 i)) v)
7 ;; #(2 4 6 8 10)
```

And I'm adding a really nice Cons gem!

```ruby
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
```
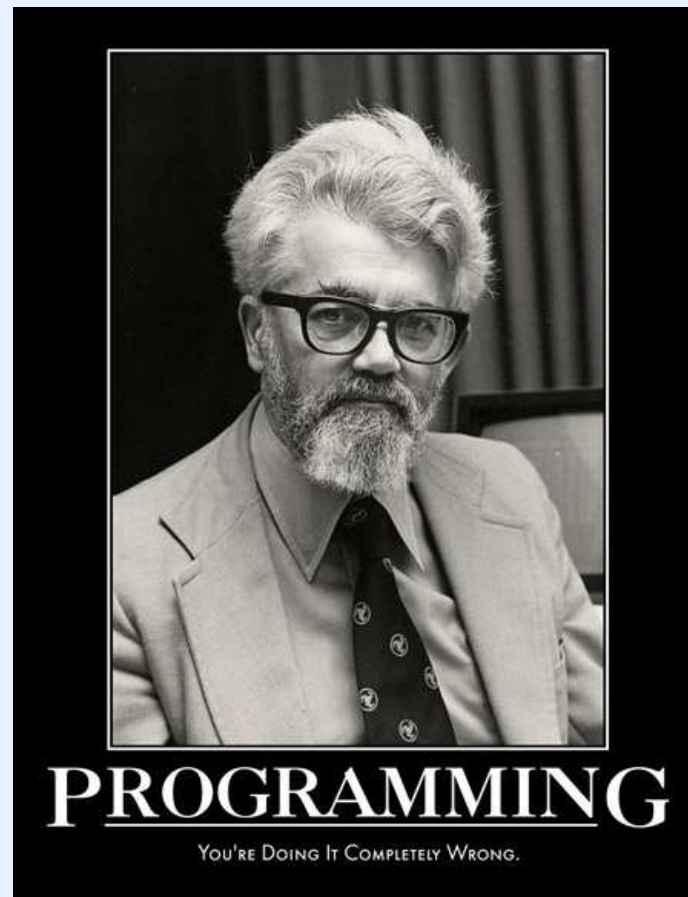
Remember linked lists?
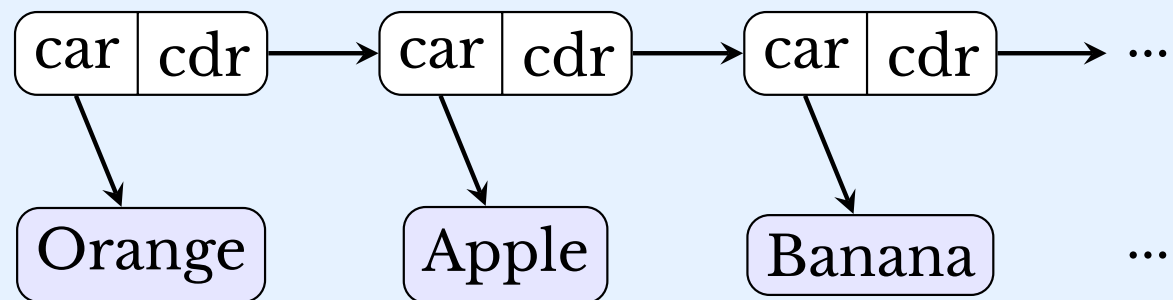

Orange →next→ Apple →next→ Banana →next→ ...

We can do this something like this in Ruby:

```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.
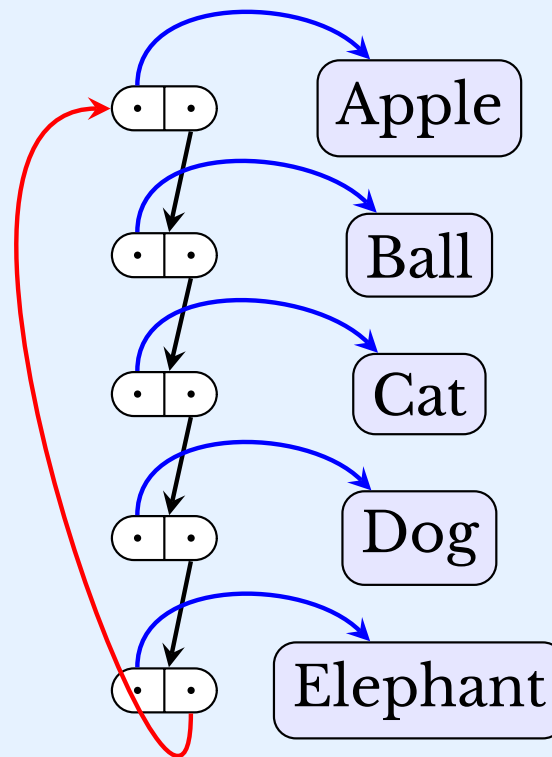## The data isn't in the cell, it's two pointers.

| car | cdr | → | car | cdr | → | car | cdr | → | ... |

Orange   Apple   Banana   ...
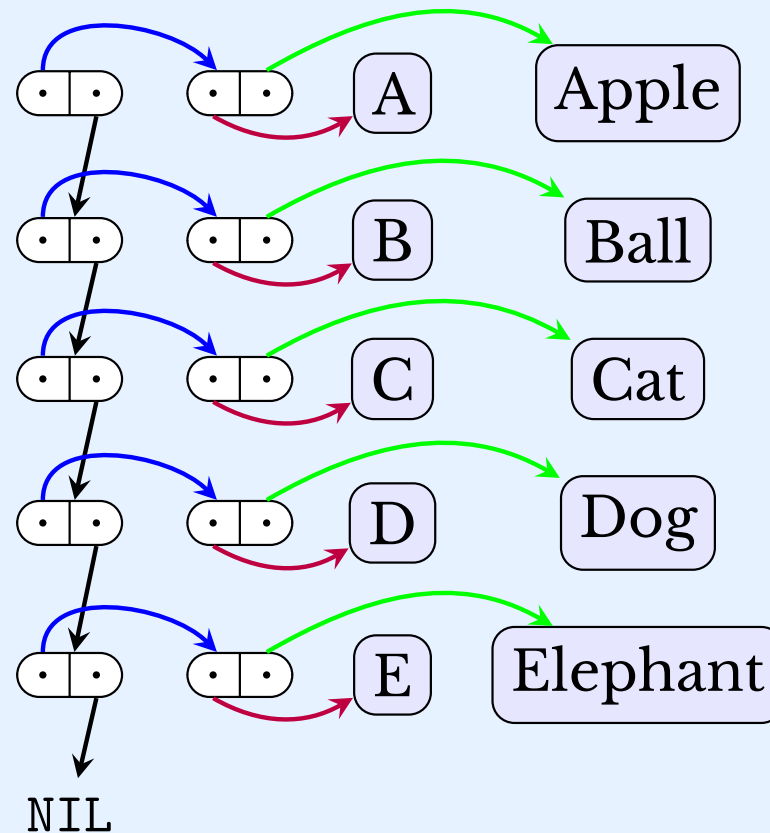
Say hello to the IBM Type 704
CAR: Contents of the Address Register
CDR: Contents of the Data Register

Why two pointers?
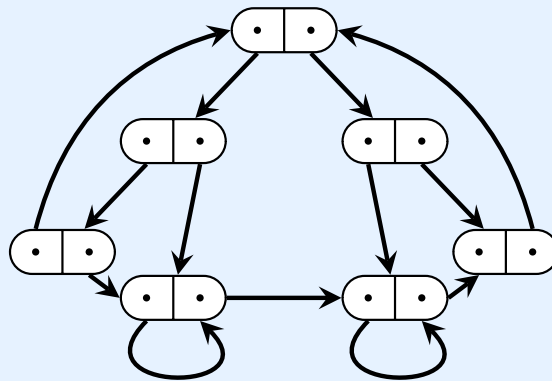# Because you can make ring buffers.

Why two pointers?
Because you can make alists.

NIL

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

```
1  '((A . Apple)
2    (B . Ball)
3    (C . Cat)
4    (D . Dog)
5    (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
1 class Cons
2   attr_accessor :car, :cdr
3   def initialize car, cdr
4     @car, @cdr = car, cdr
5     ...
6   end
7   ...
8 end
```

Chris Gore

cgore.com

Questions?