

```

1  # Ruby Classes
2  class Object
3      def evaluate(scope)
4          self
5      end
6
7      def evaluate_all(scope)
8          self
9      end
10 end
11
12 # Custom Classes
13 class PseudoCodeError < RuntimeError; end
14
15 class SuperNode
16     def initialize_global_variables(scope)
17         @@global_scope = scope
18     end
19
20     def evaluate_all(scope)
21         evaluate(scope).evaluate_all(scope)
22     end
23 end
24
25 class ProgramNode < SuperNode
26     def initialize(statements, scope)
27         @statements = statements
28         initialize_global_variables(scope)
29     end
30
31     def evaluate
32         return_value = nil
33         @statements.each do |s|
34             return_value = s.evaluate_all(@@global_scope)
35             return return_value.value if return_value.class == ReturnValue
36         end
37         return_value
38     end
39 end
40
41 class Scope
42     attr_reader :variables
43     def initialize(parent=nil)
44         @parent = parent
45         @variables = {}
46         @functions = {}
47     end
48
49     def set_var(name, value)
50         if @parent and @parent.get_var(name)
51             @parent.set_var(name, value)
52         else
53             @variables[name] = value
54         end
55     end
56
57     def get_var(name)
58         if @variables.include?(name)
59             @variables[name]
60         elsif @parent
61             @parent.get_var(name)
62         else
63             nil
64         end
65     end
66
67     def set_func(name, node)
68         @functions[name] = node
69     end
70 end

```

```

71     def get_func(name)
72         if @functions.include?(name)
73             @functions[name]
74         elsif @parent
75             @parent.get_func(name)
76         else
77             nil
78         end
79     end
80 end
81
82 class AssignmentNode < SuperNode
83     def initialize(name, value, op=nil)
84         @name, @value, @op = name, value, op
85     end
86
87     def evaluate(scope)
88         value = @value.evaluate_all(scope)
89         case @op
90         when nil then scope.set_var(@name, value)
91         when :+
92             scope.set_var(@name, AritmNode.new(
93                 scope.get_var(@name), :+, value).evaluate(scope))
94         when :-
95             scope.set_var(@name, AritmNode.new(
96                 scope.get_var(@name), :-, value).evaluate(scope))
97         when :*
98             scope.set_var(@name, AritmNode.new(
99                 scope.get_var(@name), :*, value).evaluate(scope))
100        when :/
101            scope.set_var(@name, AritmNode.new(
102                scope.get_var(@name), :/, value).evaluate(scope))
103        end
104    end
105 end
106
107 class InputNode < SuperNode
108     def initialize(var_name)
109         @name = var_name
110     end
111
112     def evaluate(scope)
113         input = gets
114         input.chomp! if input
115         AssignmentNode.new(@name, input).evaluate(scope)
116     end
117 end
118
119 class ConditionNode < SuperNode
120     def initialize(expr, stmts, elseif=nil)
121         @expression, @statements, @elseif = expr, stmts, elseif
122     end
123
124     def evaluate(parent_scope)
125         scope = Scope.new(parent_scope)
126         if not [0, false].include? @expression.evaluate(scope)
127             @statements.each do |s|
128                 return s if (s = s.evaluate(scope)).class == ReturnValue
129             end
130         elsif not [0, false, nil].include? (s = @elseif)
131             return s if (s = @elseif.evaluate(parent_scope)).class == ReturnValue
132         end
133         nil
134     end
135 end
136
137 class WhileNode < SuperNode
138     def initialize(expr, stmts)
139         @expression, @statements = expr, stmts
140     end

```

```

141
142     def evaluate(parent_scope)
143         scope = Scope.new(parent_scope)
144         while @expression.evaluate(scope)
145             @statements.each do |s|
146                 return s if (s = s.evaluate(scope)).class == ReturnValue
147             end
148         end
149         nil
150     end
151 end
152
153 class ForEachNode < SuperNode
154     def initialize(var, it, stmts)
155         @var, @iterator, @statements = var, it, stmts
156     end
157
158     def evaluate(parent_scope)
159         scope = Scope.new(parent_scope)
160         case (iterator = @iterator.evaluate(scope))
161         when ArrayNode then iterator
162         when Array then iterator
163         when String then iterator.each_char
164         else raise "Bad iterator class (#{@iterator.class}) received!"
165         end.each do |elem|
166             AssignmentNode.new(@var, elem).evaluate(scope)
167             @statements.each do |s|
168                 return s if (s = s.evaluate(scope)).class == ReturnValue
169             end
170         end
171         nil
172     end
173 end
174
175 class FromNode < SuperNode
176     def initialize(start, stop)
177         @start, @stop = start, stop
178     end
179
180     def evaluate(scope)
181         if (stop = @stop.evaluate(scope)) > (start = @start.evaluate(scope))
182             ArrayNode.new((start..stop).to_a)
183         else
184             ArrayNode.new(start.downto(stop).to_a)
185         end
186     end
187 end
188
189 class LookupNode < SuperNode
190     def initialize(name)
191         @name = name
192     end
193
194     def evaluate(scope)
195         if (results = scope.get_var(@name)).nil?
196             p scope if $DEBUG_MODE
197             raise PseudoCodeError, "Variable '#{@name}' does not exist!"
198         end
199         results
200     end
201 end
202
203 class ComparisonNode < SuperNode
204     def initialize(lh, op, rh=nil, middle=nil)
205         @lh, @op, @rh, @middle = lh, op, rh, middle
206     end
207
208     def evaluate(scope)
209         lh = @lh.evaluate(scope)
210         rh = @rh.evaluate(scope)

```

```

211     middle = @middle.evaluate(scope)
212     begin
213         case @op
214         when :not then not lh
215         when :and then lh && rh
216         when :or then lh || rh
217         when :== then lh == rh
218         when :< then lh < rh
219         when :> then lh > rh
220         when :<= then lh <= rh
221         when :>= then lh >= rh
222         when :between then middle.between?([lh,rh].min, [lh, rh].max)
223         end
224     rescue
225         if @op == :between
226             raise PseudoCodeError, "Cannot compare '#{lh}' <= #{middle} <= '#{rh}'"
227         else
228             raise PseudoCodeError, "Cannot compare '#{lh}' #{@op} '#{rh}'"
229         end
230     end
231 end
232 def set_lh(value)
233     if @op == :between
234         @middle = value
235     else
236         @lh = value
237     end
238     self
239 end
240 end
241
242 class WriteNode < SuperNode
243     def initialize(value)
244         @value = value
245     end
246
247     def evaluate(scope)
248         if $DEBUG_MODE
249             File.open("f", "a") { |f| @value.each { |a| f.print(prepare(a, scope)) } }
250         else
251             @value.each { |a| print(prepare(a, scope)) }
252         end
253         nil
254     end
255
256     def prepare(data, scope)
257         if (data = data.evaluate_all(scope)).class == String
258             data.gsub('\n', "\n").gsub('\t', "\t").gsub('\r', "\r")
259         else
260             data
261         end
262     end
263 end
264
265 class AritmNode < SuperNode
266     def initialize(lh, op, rh)
267         @lh, @op, @rh = lh, op, rh
268     end
269     def evaluate(scope)
270         lh = @lh.evaluate_all(scope)
271         rh = @rh.evaluate_all(scope)
272         begin
273             case @op
274             when :+
275                 lh.class == Array ? lh.dup << rh : lh + rh
276             when :-
277                 [Array, String].include?(lh.class) ? lh.dup[0...-rh] : lh - rh
278             when :%
279                 lh % rh
280             when :*

```

```

281         lh * rh
282     when :/
283         lh / rh
284     end
285 rescue
286     raise PseudoCodeError, "Cannot calculate #{lh} #{@op} #{rh}"
287 end
288 end
289 end
290
291 class ArrayNode < SuperNode
292     attr_reader :values
293     def initialize(values=[])
294         @values = values
295     end
296
297     def evaluate(scope)
298         @values
299     end
300
301     def evaluate_all(scope)
302         @values.map { |z| z.evaluate_all(scope) }
303     end
304
305     def <<(array)
306         @values << array
307         self
308     end
309
310     def each
311         @values.each { |e| yield(e) }
312     end
313 end
314
315 class IndexNode < SuperNode
316     def initialize(object, index)
317         @object, @index = object, index
318     end
319
320     def evaluate(scope)
321         object = @object.evaluate(scope)
322         raise PseudoCodeError, "Cannot use index on empty object" if object.empty?
323
324         index =
325             if @index == 'last' and object.methods.include?(:length)
326                 object.length
327             elsif @index.class == String
328                 @index.to_i
329             else
330                 @index.evaluate(scope)
331             end
332         if not index.class == Fixnum
333             raise PseudoCodeError, "Cannot use '#{index}' as index"
334         end
335
336         if object.methods.include?(:[])
337             if (index > 0) && (not object[index - 1].nil?)
338                 return object[index - 1]
339             else
340                 raise PseudoCodeError, "Index '#{index}' out of range"
341             end
342         end
343         raise PseudoCodeError, "Cannot use index on '#{object}'"
344     end
345 end
346
347 class LengthNode < SuperNode
348     def initialize(list)
349         @list = list
350     end

```

```

351     def evaluate(scope)
352         list = @list.evaluate_all(scope)
353         begin
354             list.length
355         rescue
356             raise PseudoCodeError, "#{list} does not have a length"
357         end
358     end
359 end
360
361 class FunctionDeclarationNode < SuperNode
362     def initialize(name, stmts, params=[])
363         @name, @parameters, @statements = name, params, stmts
364     end
365
366     def evaluate(scope)
367         @@global_scope.set_func(@name, FunctionNode.new(@parameters, @statements))
368         nil
369     end
370 end
371
372 class FunctionExecutionNode < SuperNode
373     def initialize(name, params=ArrayNode.new([]))
374         @name, @parameters = name, params
375     end
376
377     def evaluate(scope)
378         @@global_scope.get_func(@name).evaluate(
379             scope, @parameters.evaluate_all(scope)
380         )
381     end
382 end
383
384 class FunctionNode < SuperNode
385     def initialize(params, stmts)
386         @param_names, @statements = params, stmts
387     end
388
389     def evaluate(parent_scope, param_values=[])
390         if not @param_names.length == param_values.length
391             raise PseudoCodeError, "Parameter mismatch! Expected " +
392                 "#{@param_names.length}, found #{param_values.length}"
393         end
394         scope = Scope.new
395         @param_names.each_index do |i|
396             AssignmentNode.new(@param_names[i], param_values[i]).evaluate(scope)
397         end
398         @statements.each do |s|
399             return s.value if (s = s.evaluate(scope)).class == ReturnValue
400         end
401         nil
402     end
403 end
404
405 class ReturnValue
406     attr_reader :value
407     def initialize(value)
408         @value = value
409     end
410
411     def evaluate(scope)
412         ReturnValue.new(@value.evaluate_all(scope))
413     end
414 end

```