```ruby
# Ruby Classes
class Object
  def evaluate(scope)
    self
  end

  def evaluate_all(scope)
    self
  end
end

# Custom Classes
class PseudoCodeError < RuntimeError; end

class SuperNode
  def initialize_global_variables(scope)
    @@global_scope = scope
  end

  def evaluate_all(scope)
    evaluate(scope).evaluate_all(scope)
  end
end

class ProgramNode < SuperNode
  def initialize(statements, scope)
    @statements = statements
    initialize_global_variables(scope)
  end

  def evaluate
    return_value = nil
    @statements.each do |s|
      return_value = s.evaluate_all(@@global_scope)
      return return_value.value if return_value.class == ReturnValue
    end
    return_value
  end
end

class Scope
  attr_reader :variables
  def initialize(parent=nil)
    @parent = parent
    @variables = {}
    @functions = {}
  end

  def set_var(name, value)
    if @parent and @parent.get_var(name)
      @parent.set_var(name, value)
    else
      @variables[name] = value
    end
  end

  def get_var(name)
    if @variables.include?(name)
      @variables[name]
    elsif @parent
      @parent.get_var(name)
    else
      nil
    end
  end

  def set_func(name, node)
    @functions[name] = node
  end

```

```ruby
  def get_func(name)
    if @functions.include?(name)
      @functions[name]
    elsif @parent
      @parent.get_func(name)
    else
      nil
    end
  end
end

class AssignmentNode < SuperNode
  def initialize(name, value, op=nil)
    @name, @value, @op = name, value, op
  end

  def evaluate(scope)
    value = @value.evaluate_all(scope)
    case @op
    when nil then scope.set_var(@name, value)
    when :+
      scope.set_var(@name, AritmNode.new(
        scope.get_var(@name), :+, value).evaluate(scope))
    when :-
      scope.set_var(@name, AritmNode.new(
        scope.get_var(@name), :-, value).evaluate(scope))
    when :*
      scope.set_var(@name, AritmNode.new(
        scope.get_var(@name), :*, value).evaluate(scope))
    when :/
      scope.set_var(@name, AritmNode.new(
        scope.get_var(@name), :/, value).evaluate(scope))
    end
  end
end

class InputNode < SuperNode
  def initialize(var_name)
    @name = var_name
  end

  def evaluate(scope)
    input = gets
    input.chomp! if input
    AssignmentNode.new(@name, input).evaluate(scope)
  end
end

class ConditionNode < SuperNode
  def initialize(expr, stmts, elseif=nil)
    @expression, @statements, @elseif = expr, stmts, elseif
  end

  def evaluate(parent_scope)
    scope = Scope.new(parent_scope)
    if not [0, false].include? @expression.evaluate(scope)
      @statements.each do |s|
        return s if (s = s.evaluate(scope)).class == ReturnValue
      end
    elsif not [0, false, nil].include? (s = @elseif)
      return s if (s = @elseif.evaluate(parent_scope)).class == ReturnValue
    end
    nil
  end
end

class WhileNode < SuperNode
  def initialize(expr, stmts)
    @expression, @statements = expr, stmts
  end
```

```ruby

  def evaluate(parent_scope)
    scope = Scope.new(parent_scope)
    while @expression.evaluate(scope)
      @statements.each do |s|
        return s if (s = s.evaluate(scope)).class == ReturnValue
      end
    end
    nil
  end
end

class ForEachNode < SuperNode
  def initialize(var, it, stmts)
    @var, @iterator, @statements = var, it, stmts
  end

  def evaluate(parent_scope)
    scope = Scope.new(parent_scope)
    case (iterator = @iterator.evaluate(scope))
    when ArrayNode then iterator
    when Array then iterator
    when String then iterator.each_char
    else raise "Bad iterator class (#{iterator.class}) received!"
    end.each do |elem|
      AssignmentNode.new(@var, elem).evaluate(scope)
      @statements.each do |s|
        return s if (s = s.evaluate(scope)).class == ReturnValue
      end
    end
    nil
  end
end

class FromNode < SuperNode
  def initialize(start, stop)
    @start, @stop = start, stop
  end

  def evaluate(scope)
    if (stop = @stop.evaluate(scope)) > (start = @start.evaluate(scope))
      ArrayNode.new((start..stop).to_a)
    else
      ArrayNode.new(start.downto(stop).to_a)
    end
  end
end

class LookupNode < SuperNode
  def initialize(name)
    @name = name
  end

  def evaluate(scope)
    if (results = scope.get_var(@name)).nil?
      p scope if $DEBUG_MODE
      raise PseudoCodeError, "Variable '#{@name}' does not exist!"
    end
    results
  end
end

class ComparisonNode < SuperNode
  def initialize(lh, op, rh=nil, middle=nil)
    @lh, @op, @rh, @middle = lh, op, rh, middle
  end

  def evaluate(scope)
    lh = @lh.evaluate(scope)
    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
```

```ruby
  def evaluate(scope)
    list = @list.evaluate_all(scope)
    begin
      list.length
    rescue
      raise PseudoCodeError, "#{list} does not have a length"
    end
  end
end

class FunctionDeclarationNode < SuperNode
  def initialize(name, stmts, params=[])
    @name, @parameters, @statements = name, params, stmts
  end

  def evaluate(scope)
    @@global_scope.set_func(@name, FunctionNode.new(@parameters, @statements))
    nil
  end
end

class FunctionExecutionNode < SuperNode
  def initialize(name, params=ArrayNode.new([]))
    @name, @parameters = name, params
  end

  def evaluate(scope)
    @@global_scope.get_func(@name).evaluate(
      scope, @parameters.evaluate_all(scope)
    )
  end
end

class FunctionNode < SuperNode
  def initialize(params, stmts)
    @param_names, @statements = params, stmts
  end

  def evaluate(parent_scope, param_values=[])
    if not @param_names.length == param_values.length
      raise PseudoCodeError, "Parameter mismatch! Expected " +
        "#{@param_names.length}, found #{param_values.length}"
    end
    scope = Scope.new
    @param_names.each_index do |i|
      AssignmentNode.new(@param_names[i], param_values[i]).evaluate(scope)
    end
    @statements.each do |s|
      return s.value if (s = s.evaluate(scope)).class == ReturnValue
    end
    nil
  end
end

class ReturnValue
  attr_reader :value
  def initialize(value)
    @value = value
  end

  def evaluate(scope)
    ReturnValue.new(@value.evaluate_all(scope))
  end
end
```