```ruby
#! /usr/bin/env ruby
# -*- coding: utf-8 -*-

import_path =
  if File.symlink?(__FILE__)
    File.dirname(File.absolute_path(File.readlink(__FILE__)))
  else
    File.dirname(File.absolute_path(__FILE__))
  end

require "#{import_path}/rdparse.rb"
require "#{import_path}/nodes.rb"

class PseudoCode
  def initialize(scope=Scope.new)
    @parser = Parser.new("pseudo parser") do
      token(/#.*?$/)             # Comments
      token(/".*?"/)             { |m| m.to_s } # Strings
      token(/(-?\d+)*[a-zA-Z]+/) { |m| m }      # Variables, keywords, etc
      token(/-?\d+\.\d+/)        { |m| m.to_f } # Floats
      token(/-?\d+/)             { |m| m.to_i } # Integers
      token(/\n+/)               # :newline, :indent and :dedent tokens
      token(/[^ ]/)              { |m| m } # Non-space characters
      token(/ /)                 # Throw away spaces

      start(:program) do
        match(:top_level_statements) { |a| ProgramNode.new(a, scope).evaluate }
      end

      # Statements only allowed in the global scope
      rule(:top_level_statements) do
        match(:top_level_statements, :func_decl) { |a, b| a << b }
        match(:top_level_statements, :statements) { |a, b| a + b }
        match(:prompt, :prompt_rules) { |_, a| a }
        match(:empty) { [] }
      end

      rule(:prompt_rules) do
        match(:expression, :newline) { |a, _| [a] }
        match(:statements) { |a| a }
      end

      # Statements allowed in any scope
      rule(:statements) do
        match(:newline, :statements) { |_, a| a }
        match(:statements, :newline, :statement) { |a, _, b| a << b }
        match(:statements, :newline) { |a, _| a }
        match(:statement) { |a| [a] }
      end

      rule(:statement) do
        match('write', :expression_list) { |_, a| WriteNode.new(a) }
        match('read', 'to', :identifier) { |_, _, a| InputNode.new(a) }
        match(:condition)
        match(:while)
        match(:foreach)
        match('return', :expression) { |_, a| ReturnValue.new(a) }
        match(:assignment)
        match(:func_exec)
        match(:newline)
      end

      rule(:condition) do
        match('if', :expression, 'then', :newline,
              :indent, :statements, :dedent, :condition_else) {
          |_, if_expr, _, _, _, if_stmts, _, elseif|
          ConditionNode.new(if_expr, if_stmts, elseif) }
      end

      rule(:condition_else) do
```

```
71          match(:newline, 'else', 'if', :expression, 'then', :newline,
72                :indent, :statements, :dedent, :condition_else) {
73            |_, _, _, if_expr, _, _, _, if_stmts, _, elseif|
74            ConditionNode.new(if_expr, if_stmts, elseif) }
75          match(:newline, 'else', :newline, :indent, :statements, :dedent) {
76            |_, _, _, _, stmts, _| ConditionNode.new(true, stmts) }
77          match(:empty)
78        end
79
80        rule(:while) do
81          match('while', :expression, 'do', :newline,
82                :indent, :statements, :dedent) { |_, expr, _, _ , _, stmts, _|
83            WhileNode.new(expr, stmts) }
84        end
85
86        rule(:foreach) do
87          match('for', 'each', :identifier, :foreach_list, 'do', :newline,
88                :indent, :statements, :dedent) {
89            |_, _, var, iterator, _, _, _, stmts, _|
90            ForEachNode.new(var, iterator, stmts) }
91        end
92
93        rule(:assignment) do
94          match(:identifier, 'equals', :expression) { |lh, _, rh|
95            AssignmentNode.new(lh, rh) }
96          match(:assign_mod, :identifier, 'by', :expression) { |mod, lh, _, rh|
97            AssignmentNode.new(lh, rh, mod) }
98        end
99
100       rule(:expression) do
101         match(:expression, :and_or, :expression) { |lh, sym, rh|
102           ComparisonNode.new(lh, sym, rh) }
103         match('not', :expression) { |sym, e| ComparisonNode.new(e, sym.to_sym) }
104         match(:bool)
105         match(:comparison)
106       end
107
108       rule(:comparison) do
109         match(:comparable, 'is', :comparison_tail) { |e, _, comp_node|
110           comp_node.set_lh(e) }
111         match(:aritm_expr)
112       end
113
114       rule(:comparison_tail) do
115         match('less', 'than', :comparable) do |_, _, e|
116           ComparisonNode.new(nil, :<, e); end
117         match('greater', 'than', :comparable) do  |_, _, e|
118           ComparisonNode.new(nil, :>, e); end
119         match(:comparable, 'or', 'more') do |e, _, _|
120           ComparisonNode.new(nil, :>=, e); end
121         match(:comparable, 'or', 'less') do |e, _, _|
122           ComparisonNode.new(nil, :<=, e); end
123         match('between', :comparable, 'and', :comparable) do |_, e, _, f|
124           ComparisonNode.new(e, :between, f, nil); end
125         match(:comparable) { |e| ComparisonNode.new(nil, :==, e) }
126       end
127
128       rule(:aritm_expr) do
129         match(:aritm_expr, :plus_minus, :term) { |lh, mod, rh|
130           AritmNode.new(lh, mod, rh) }
131         match(:term)
132       end
133
134       rule(:term) do
135         match(:term, :mult_div, :factor) { |lh, mod, rh|
136           AritmNode.new(lh, mod, rh) }
137         match(:factor)
138       end
139
140       rule(:factor) do
```

```
141           match(:factor, 'modulo', :factor) { |a, _, b| AritmNode.new(a, :%, b) }
142           match('(', :expression, ')') { |_, m, _| m }
143           match(:float)
144           match(:integer)
145           match(:func_exec)
146           match(:index, 'of', :indexable) do |index, _, list|
147             IndexNode.new(list, index); end
148           match('size', 'of', :indexable) { |_, _, list| LengthNode.new(list) }
149           match(:variable_get)
150           match(:string)
151           match(:array)
152         end
153
154         rule(:func_decl) do
155           match(:identifier, 'with', :identifier_list, 'does', :newline,
156                 :indent, :statements, :dedent) do
157             |name, _, params, _, _, _, stmts, _|
158             FunctionDeclarationNode.new(name, stmts, params); end
159           match(:identifier, 'does', :newline,
160                 :indent, :statements, :dedent) do |name, _, _, _, stmts, _|
161             FunctionDeclarationNode.new(name, stmts); end
162         end
163
164         rule(:func_exec) do
165           match('do', :identifier, 'with', :expression_list) do
166             |_, name, _, params|
167             FunctionExecutionNode.new(name, params); end
168           match('do', :identifier) { |_, name| FunctionExecutionNode.new(name) }
169         end
170
171         # Lists
172         rule(:identifier_list) do
173           match(:identifier_list, ',', :identifier) { |a, _, b| a << b }
174           match(:identifier) { |m| [m] }
175         end
176
177         rule(:expression_list) do
178           match(:expression_list, ',', :expression) { |a, _, b| a << b }
179           match(:expression) { |m| ArrayNode.new([m]) }
180         end
181
182         rule(:foreach_list) do
183           match('in', :expression) { |_, iterator| iterator }
184           match('from', :foreach_elem, 'to', :foreach_elem) do |_, start, _, stop|
185             FromNode.new(start, stop); end
186         end
187
188         # Collections
189           rule(:foreach_elem) do
190           match(:variable_get)
191           match(:integer)
192         end
193
194         rule(:assign_mod) do
195           match('increase') { :+ }
196           match('decrease') { :- }
197           match('multiply') { :* }
198           match('divide') { :/ }
199         end
200
201         rule(:and_or) do
202           match('and') { :and }
203           match('or')  { :or  }
204         end
205
206         rule(:plus_minus) do
207           match('plus') { :+ }
208           match('minus') { :- }
209         end
210
```

```
211         rule(:mult_div) do
212           match('times') { :* }
213           match('divided', 'by') { :/ }
214         end
215
216         rule(:comparable) do
217           match(:aritm_expr)
218           match(:string)
219           match(:array)
220         end
221
222         rule(:index) do
223           match(/^\d*(11|12|13)th$/)
224           match(/^\d*1st$/)
225           match(/^\d*2nd$/)
226           match(/^\d*3rd$/)
227           match(/^\d+th$/)
228           match(/^[a-zA-Z]+th$/) { |m| LookupNode.new(m[0...-2]) }
229           match('last')
230         end
231
232         rule(:indexable) do
233           match(:string)
234           match(:array)
235           match(:variable_get)
236         end
237
238         # Types
239         rule(:float)        { match(Float) }
240         rule(:integer)      { match(Integer) }
241         rule(:identifier)   { match(/^[a-zA-Z]+$/) }
242         rule(:variable_get) { match(:identifier) { |m| LookupNode.new(m) } }
243         rule(:string)       { match(/".*"/) { |m| m.delete('"') } }
244         rule(:array) do
245           match('[', :expression_list, ']') { |_, m, _| m }
246           match('[', ']') { ArrayNode.new }
247         end
248         rule(:bool) do
249           match('true') { true }
250           match('false') { false }
251         end
252       end
253     end
254
255     def parse(str, interactive=false)
256       str = str + "\n"
257       if $DEBUG_MODE
258         @parser.parse(str, interactive)
259       else
260         begin
261           @parser.parse(str, interactive)
262         rescue Parser::ParseError => e
263           $stderr.puts "SyntaxError: #{e}"
264         rescue => e
265           $stderr.puts "#{e.class}: #{e}"
266         end
267       end
268     end
269
270     def prompt
271       require 'readline'
272       log(false)
273       begin
274         while input = Readline.readline(">> ", true)
275           break if input == "exit"
276           return_value = parse(input, true)
277           p return_value unless return_value.nil?
278         end
279       rescue Interrupt
280         puts
```

```ruby
        exit
      end
    end

    def log(state = true)
      @parser.logger.level = state ? Logger::DEBUG : Logger::WARN
    end
  end

  if __FILE__ == $0
    pc = PseudoCode.new
    pc.log(false)

    # If no argument, parse either stdin-data or start a prompt
    if ARGV.empty?
      if $stdin.tty?
        pc.prompt
      else
        pc.parse $stdin.read
      end

      # Otherwise, try to parse file
    else
      parse_data =
        begin
          File.read(ARGV[0])
        rescue SystemCallError => e
          $stderr.puts e
          exit 1
        end
      pc.parse(parse_data)
    end
  end
```