William Setzer
a02 Ruby
24 February 2018

Using the code provided by the TA in the lab, I was able to fill in the rest of the rules. This was one of the hardest projects I've done for a class, partially because this was the first time I have written a lexer, but mostly because I have never written in Ruby before and a lexer is a complicated program. To help with the design, I use the dragon book by Aho, which writes a lexer early on in the book. This helped me to understand lexers better, but made Ruby no less difficult.

I do not know what I liked about Ruby. The boolean operators are interesting (calling dot methods on objects). However, I disliked most everything else. The syntax and style rules led to long, confusing if-else blocks, which hurts the readability of the program. Honestly, when it comes to scripting languages, I would pick Python every time.

LEXER

```
#!/usr/bin/ruby -w

require_relative 'parser.rb'

class Lexer
   # lexer class

   # attr_accessor :count
   attr_accessor :prog_text
   attr_accessor :queue

   def initialize(program_file_name)
       #instance variables
       @prog_text = ''

       open(program_file_name) {|f|
         data = f.read
         @prog_text += data
       }

       @queue = Array.new #calls the Queueclass

   end
```

```
    def identifier?
        if(((/[[a-z]|[A-Z]][\w]*/ =~ @prog_text[0..-1]) == 0) &&
!((/if|while|not|end|do|true|   false|and/ =~ @prog_text[0..-1]) == 0))
            identifier = /[[a-z]|[A-Z]][\w]*/.match(@prog_text[0..-1])[0]
            @prog_text.sub!(identifier, "")
            #push the identifier to the queue
            @queue.push [identifier,"identifier"]
            return TRUE
        else
            return FALSE
        end
    end
    def keyword?
        if((/if|while|not|end|do|and/ =~ @prog_text[0..-1]) == 0)
            keyword = /if|while|not|end|do|and/.match(@prog_text[0..-1])[0]
            @prog_text.sub!(keyword, "")
            #push the keyword to the queue
            @queue.push [keyword,"keyword"]
            return TRUE
        else
            return FALSE
        end
    end


    def integer?
        if((/[0-9]+/ =~ @prog_text[0..-1]) == 0)
            intg = /[0-9]+/.match(@prog_text[0..-1])[0]
            @prog_text.sub!(intg, "")
            #push the integer to the queue
            @queue.push [intg,"integer"]
            return TRUE
        else
            return FALSE
        end
    end
    def boolean?
        if((/true|false/ =~ @prog_text[0..-1]) == 0)
            bool = /true|false/.match(@prog_text[0..-1])[0]
            @prog_text.sub!(bool, "")
            #push the boolean to the queue
            @queue.push [bool,"boolean"]
            return TRUE
        else
            return FALSE
        end
    end
    def operator?
        if((/\+|-|\*|\/|:=|<=|>=|>|</ =~ @prog_text[0..-1]) == 0)
            operator = /\+|-|\*|\/|:=|<=|>=|>|</.match(@prog_text[0..-1])[0]
            @prog_text.sub!(operator, "")
            #push the operator to the queue
            @queue.push [operator,"operator"]
            return TRUE
        else
            return FALSE
        end
    end
    def comment?
        if((/\/\/.*/ =~ @prog_text[0..-1]) == 0)
            comment = /\/\/.*/.match(@prog_text[0..-1])[0]
            @prog_text.sub!(comment, "")
            return TRUE
        else
            return FALSE
        end
    end


    def seperator?
        if((/\(|\)|;/ =~ @prog_text[0..-1]) == 0)
            seperator = /\(|\)|;/.match(@prog_text[0..-1])[0]
            @prog_text.sub!(seperator, "")
            #push the operator to the queue
            @queue.push [seperator,"seperator"]
            return TRUE
        else
            return FALSE
        end
    end


    def space?
        if((/\s/ =~ @prog_text[0..-1]) == 0)
```

```
           spaces = /\s/.match(@prog_text[0..-1])[0]
           @prog_text.sub!(spaces, "")
           return TRUE
       else
           return FALSE
       end
   end
   def run
       while @prog_text
           if self.identifier? | self.keyword? | self.integer? | self.boolean? |
self.comment? | self.operator? | self.seperator? | self.space?
               #puts @queue[-1][0]
           else
               #puts "error at: "
               puts @prog_text[0..-1]
               break
           end
       end
       @queue.push ["EOF","EOF"]
       #puts @queue
   end

end

# main program
program_name = ARGV[0]
if program_name.nil?
   puts "Missing program name!"
   exit!
end
lexer = Lexer.new(program_name)
# a = Lexer.new('prog.txt')
lexer.run

parser = Parser.new(lexer.queue)
puts parser.program?
```

===
PARSER

```ruby
#!/usr/bin/ruby -w

class Parser

   def initialize(token_queue)
       @tok_queue = token_queue.reverse
       @temp_stack = Array.new
       @err_queue = Array.new
   end


   def getTokenKind
       # if the token queue is not empty,
       if ! @tok_queue.empty?
           return @tok_queue[-1][1]
       else
           return "empty"
       end
   end


   def clear_err_queue
       @err_queue.clear
   end


   def update_err_queue
       puts @err_queue[0]
       @err_queue.insert(0,@tok_queue[-1])
   end


   def getTokenText
       if ! @tok_queue.empty?
           return @tok_queue[-1][0]
       else
           return "empty"
       end
   end


   def nextToken
```

```
    if ! @tok_queue.empty?
        @temp_stack.push(@tok_queue.pop)
    else
        puts "empty"
    end
end


def backtrack(top)
    while (@tok_queue[-1] != top) && (not @tok_queue.empty?) do
        @tok_queue.push(@temp_stack.pop)
    end
end


def program?
    #puts @tok_queue
    #puts " "
    if self.stmts?
        #puts @tok_queue
        puts " "
        if self.getTokenText == 'EOF'
            self.nextToken
            puts "No errors"
            return TRUE
        else
            puts "Syntax error"
            self.update_err_queue
            puts @err_queue[0]
            return FALSE
        end
    else
        puts "Syntax error"
        self.update_err_queue
        puts @err_queue[0]
        return FALSE
    end
end


def stmts?
    top = @tok_queue[-1]
    if self.stmt?
        #puts "statment"
        if self.getTokenText == ';'
            self.nextToken
            while self.stmt?
                #puts @tok_queue
                if self.getTokenText == ';'
                    self.nextToken
                else
                    break
                end
            end

            self.clear_err_queue
            return TRUE
        else
            self.update_err_queue
            self.backtrack(top)
            return FALSE
        end
    else
        self.update_err_queue
        self.backtrack(top)
        return FALSE
    end
end


def stmt?
    top = @tok_queue[-1]
    if self.getTokenKind == "identifier"
        self.nextToken
        if self.getTokenText == ":="
            self.nextToken
            if self.addop?
                self.clear_err_queue
                return TRUE
            else
                self.update_err_queue
                #backtrack
                self.backtrack(top)
                return FALSE
            end
        else
            self.update_err_queue
            self.backtrack(top)
            return FALSE
        end

    elsif self.getTokenText == "if"
        self.nextToken
        if self.lexpr?
            if self.getTokenText == "then"
                self.nextToken
                if self.stmts?
                    if self.getTokenText == "else"
                        self.nextToken
                        if self.stmts?
                            if self.getTokenText == "end"
                                self.nextToken
                                self.clear_err_queue
                                return TRUE
                            else
                                self.update_err_queue
                                self.backtrack(top)
                                return FALSE
                            end
                        else
                            self.update_err_queue
                            self.backtrack(top)
                            return FALSE
                        end
                    else
                        self.update_err_queue
                        self.backtrack(top)
                        return FALSE
                    end
                else
                    self.update_err_queue
                    self.backtrack(top)
                    return FALSE
                end
            else
                self.update_err_queue
```

```
                    self.backtrack(top)
                    return FALSE
                end
            else
                self.update_err_queue
                self.backtrack(top)
                return FALSE
            end

        elsif self.getTokenText == "while"
            self.nextToken
            if self.lexpr?
                if self.getTokenText == "do"
                    self.nextToken
                    if self.stmts?
                        if self.getTokenText == "end"
                            self.nextToken
                            self.clear_err_queue
                            return TRUE
                        else
                            self.update_err_queue
                            self.backtrack(top)
                            return FALSE
                        end
                    else
                        self.update_err_queue
                        self.backtrack(top)
                        return FALSE
                    end
                else
                    self.update_err_queue
                    self.backtrack(top)
                    return FALSE
                end
            else
                self.update_err_queue
                self.backtrack(top)
                return FALSE
            end

        else
            self.update_err_queue
            self.backtrack(top)
            return FALSE
        end
    end


    def addop?
        top = @tok_queue[-1]
        if self.mulop?
            if (self.getTokenText == "+") | (self.getTokenText == "-")
                self.nextToken
                if self.addop?
                    self.clear_err_queue
                    return TRUE
                else
                    self.update_err_queue
                    #backtrack
                    self.backtrack(top)
                    return FALSE
                end
            else
                self.clear_err_queue
                return TRUE
            end
        else
            self.update_err_queue
            self.backtrack(top)
            return FALSE
        end
    end


    def mulop?
        top = @tok_queue[-1]
        if self.factor?
            if (self.getTokenText == "/") | (self.getTokenText == "*")
                self.nextToken
                if self.mulop?
                    self.clear_err_queue
                    return TRUE
                else
                    #backtrack
```

```
                    self.update_err_queue
                    self.backtrack(top)
                    return FALSE
                end
            else
                self.clear_err_queue
                return TRUE
            end
        else
            self.update_err_queue
            self.backtrack(top)
            return FALSE
        end
    end


    def factor?
        top = @tok_queue[-1]
        if self.getTokenKind == "integer"
            self.nextToken
            self.clear_err_queue
            return TRUE

        elsif self.getTokenKind == "identifier"
            self.nextToken
            self.clear_err_queue
            return TRUE
        #tests for ( <addop> )
        elsif self.getTokenText == '('
            self.nextToken
            if self.addop?
                if self.getTokenText == ')'
                    self.nextToken
                    self.clear_err_queue
                    return TRUE
                else
                    self.update_err_queue
                    #backtrack
                    self.backtrack(top)
                    return FALSE
                end
            else
                self.update_err_queue
                self.backtrack(top)
                return FALSE
            end
        else
            self.update_err_queue
            self.backtrack(top)
            return FALSE
        end
    end


    def lexpr?
        top = @tok_queue[-1]
        if lterm?
            if self.getTokenText == 'and'
                self.nextToken
                if self.lexpr?
                    self.clear_err_queue
                    return TRUE
                else
                    self.update_err_queue
                    #backtrack
                    self.backtrack(top)
                    return FALSE
                end
            else
                self.clear_err_queue
                return TRUE
            end
        else
            self.clear_err_queue
            return TRUE
        end
    end


    def lterm?
        top = @tok_queue[-1]
        if self.getTokenText == "not"
            self.nextToken
            if self.lfactor?
```

```
                self.clear_err_queue
                return TRUE
            else
                self.update_err_queue
                #backtrack
                self.backtrack(top)
                return FALSE
            end
        elsif self.lfactor?
            self.clear_err_queue
            return TRUE
        else
            self.update_err_queue
            self.backtrack(top)
            return FALSE
        end
    end


    def lfactor?
        if self.getTokenText == "true"
            self.nextToken
            self.clear_err_queue
            return TRUE
        elsif self.getTokenText == "false"
            self.nextToken
            self.clear_err_queue
            return TRUE
        elsif self.relop?
            self.clear_err_queue
            return TRUE
        else
            self.update_err_queue
            return FALSE
        end
    end


    def relop?
        top = @tok_queue[-1]
        if self.addop?
            if self.getTokenText == "<=" || self.getTokenText == "<" ||
self.getTokenText == "="
               self.nextToken
               if self.addop?
                   self.clear_err_queue
                   return TRUE
               else
                   self.update_err_queue
                   #backtrack
                   self.backtrack(top)
                   return FALSE
               end
           else
               self.clear_err_queue
               return TRUE
           end
       else
           self.update_err_queue
           self.backtrack(top)
           return FALSE
       end
   end
end
```