Shork#

Miss Ylva Llywelyn

2023/10/14

Contents

GRAMMAR

This is a notation for writing down the grammar of the language. It uses regex syntax, with the components themselves being italicised.

statements	NEWLINE* *statement* (NEWLINE+ *statement*)* NEWLINE*
statement	KEYWORD:VAR IDENTIFIER = *expression* KEYWORD:CONTINUE KEYWORD:BREAK *expression*
expression	

Code Listing

Listing 1: Lexer.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Globalization;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace ShorkSharp
9  {
10     public class Lexer
11     {
12         static readonly string[] KEYWORDS =
13         {
14             "var",
15             "func",
16             "while",
17             "do",
18             "if",
19             "then",
20             "elif",
21             "else"
22         };
23         static readonly char[] WHITESPACE = { ' ', '\t', '\r' };
24         static readonly char[] DIGITS = { '0', '1', '2', '3', '4', '5', '6',
               ↪ '7', '8', '9' };
25         static readonly char[] DIGITS_WITH_DOT = DIGITS.Concat(new char[] { '.'
               ↪ }).ToArray();
26         static readonly char[] LETTERS = { 'a', 'b', 'c', 'd', 'e', 'f', 'g',
               ↪ 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
               ↪ 'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
               ↪ 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
               ↪ 'U', 'V', 'W', 'X', 'Y', 'Z' };
27         static readonly char[] LETTERS_WITH_UNDERSCORE = LETTERS.Concat(new
               ↪ char[] { '_' }).ToArray();
28
29         public Position position { get; protected set; }
30         public string input { get; protected set; }
31         public char currentChar { get; protected set; } = '\0';
32
33         public Lexer(string input)
34         {
35             this.input = input;
36             this.position = new Position(input);
37         }
38         public Lexer(string input, string filename)
39         {
40             this.input = input;
41             this.position = new Position(filename);
42         }
43
44         void Advance()
45         {
46             position.Advance(currentChar);
47
48             if (position.index < input.Length)
49                 currentChar = input[position.index];
50             else
51                 currentChar = '\0';
52         }
```

```
53
54          public (Token[], ShorkError?) Lex()
55          {
56              if (input.Length == 0)
57                  return (new Token[] { }, new ShorkError("Empty Input", "Input
                        ↪ text is empty", null));
58              this.currentChar = input[0];
59
60              List<Token> tokens = new List<Token>();
61
62              while (currentChar != '\0')
63              {
64                  if (WHITESPACE.Contains(currentChar))
65                  {
66                      Advance();
67                  }
68
69                  // Number Tokens
70                  else if (DIGITS.Contains(currentChar))
71                  {
72                      tokens.Add(MakeNumberToken());
73                  }
74
75                  // String Tokens
76                  else if (currentChar == '"')
77                  {
78                      (Token token, ShorkError error) = MakeStringToken();
79                      if (error != null)
80                          return (null, error);
81                      tokens.Add(token);
82                  }
83
84                  // Identifiers and Keywords
85                  else if (LETTERS.Contains(currentChar))
86                  {
87                      tokens.Add(MakeIdentifierToken());
88                  }
89
90                  // Simple tokens
91                  else
92                  {
93                      switch (currentChar)
94                      {
95                          default:
96                              return (new Token[] { },
97                                      new
                                            ↪ InvalidCharacterError(string.Format("'{0}'",
                                            ↪ currentChar), position));
98                          case '+':
99                              tokens.Add(new Token(TokenType.PLUS, position));
100                             Advance();
101                             break;
102                         case '-':
103                             tokens.Add(new Token(TokenType.MINUS, position));
104                             Advance();
105                             break;
106                         case '*':
107                             tokens.Add(new Token(TokenType.MULTIPLY, position));
108                             Advance();
109                             break;
110                         case '/':
111                             tokens.Add(new Token(TokenType.DIVIDE, position));
```

```
112                             Advance();
113                             break;
114                         case '^':
115                             tokens.Add(new Token(TokenType.EXPONENT, position));
116                             Advance();
117                             break;
118
119                         case '!':
120                             (Token token, ShorkError error) =
                                    ↪ MakeNotEqualsToken();
121                             if (error != null) return (null, error);
122                             tokens.Add(token);
123                             break;
124                         case '=':
125                             tokens.Add(MakeEqualsToken());
126                             break;
127                         case '<':
128                             tokens.Add(MakeLessThanToken());
129                             break;
130                         case '>':
131                             tokens.Add(MakeGreaterThanToken());
132                             break;
133
134                         case '.':
135                             tokens.Add(new Token(TokenType.DOT, position));
136                             Advance();
137                             break;
138                         case ',':
139                             tokens.Add(new Token(TokenType.COMMA, position));
140                             Advance();
141                             break;
142
143                         case '(':
144                             tokens.Add(new Token(TokenType.LPAREN, position));
145                             Advance();
146                             break;
147                         case ')':
148                             tokens.Add(new Token(TokenType.RPAREN, position));
149                             Advance();
150                             break;
151                         case '{':
152                             tokens.Add(new Token(TokenType.LBRACE, position));
153                             Advance();
154                             break;
155                         case '}':
156                             tokens.Add(new Token(TokenType.RBRACE, position));
157                             Advance();
158                             break;
159                         case '[':
160                             tokens.Add(new Token(TokenType.LBRACKET, position));
161                             Advance();
162                             break;
163                         case ']':
164                             tokens.Add(new Token(TokenType.RBRACKET, position));
165                             Advance();
166                             break;
167                     }
168                 }
169             }
170
171             return (tokens.ToArray(), null);
172         }
```

```
173
174         Token MakeNumberToken()
175         {
176             string numstring = string.Empty + currentChar;
177             bool hasDecimalPoint = false;
178             Position startPosition = position.Copy();
179
180             Advance();
181             while (DIGITS_WITH_DOT.Contains(currentChar))
182             {
183                 if (currentChar == '.')
184                 {
185                     if (hasDecimalPoint)
186                         break;
187                     else
188                         hasDecimalPoint = true;
189                 }
190                 numstring += currentChar;
191                 Advance();
192             }
193
194             return new Token(TokenType.NUMBER, decimal.Parse(numstring),
                    ↪ startPosition, position);
195         }
196
197         (Token, ShorkError) MakeStringToken()
198         {
199             Position startPosition = position.Copy();
200             string str = string.Empty;
201             Advance();
202
203             bool escaping = false;
204             while (true)
205             {
206                 if (escaping)
207                 {
208                     switch (currentChar)
209                     {
210                         default:
211                             return (null, new
                                    ↪ InvalidEscapeSequenceError(string.Format("\\{0}",
                                    ↪ currentChar), position));
212                         case '"':
213                             str += '"';
214                             break;
215                         case '\\':
216                             str += '\\';
217                             break;
218                         case 't':
219                             str += '\t';
220                             break;
221                     }
222                     escaping = false;
223                 }
224
225                 else if (currentChar == '"')
226                 {
227                     Advance();
228                     break;
229                 }
230
231                 else if (currentChar == '\\')
```

```
232                     escaping = true;
233
234                 else
235                     str += currentChar;
236
237                 Advance();
238             }
239
240             return (new Token(TokenType.STRING, str, startPosition, position),
                    ↪ null);
241         }
242
243         Token MakeIdentifierToken()
244         {
245             Position startPosition = position.Copy();
246             string idstr = string.Empty + currentChar;
247             Advance();
248
249             while (LETTERS_WITH_UNDERSCORE.Contains(currentChar))
250             {
251                 idstr += currentChar;
252                 Advance();
253             }
254
255             if (idstr == "true")
256                 return new Token(TokenType.BOOL, true, startPosition, position);
257             else if (idstr == "false")
258                 return new Token(TokenType.BOOL, false, startPosition, position);
259             else if (idstr == "null")
260                 return new Token(TokenType.NULL, startPosition, position);
261             else
262             {
263                 TokenType ttype = KEYWORDS.Contains(idstr.ToLower()) ?
                        ↪ TokenType.KEYWORD : TokenType.IDENTIFIER;
264                 return new Token(ttype, idstr, startPosition, position);
265             }
266         }
267
268         Token MakeEqualsToken()
269         {
270             Position startPosition = position.Copy();
271             TokenType ttype = TokenType.EQUALS;
272             Advance();
273             if (currentChar == '=')
274             {
275                 ttype = TokenType.DOUBLE_EQUALS;
276                 Advance();
277             }
278             return new Token(ttype, startPosition, position);
279         }
280
281         (Token, ShorkError) MakeNotEqualsToken()
282         {
283             Position startPosition = position.Copy();
284             Advance();
285             if (currentChar == '=')
286             {
287                 Advance();
288                 return (new Token(TokenType.NOT_EQUALS, startPosition,
                        ↪ position), null);
289             }
290             return (null, new InvalidCharacterError("", position));
```

```
291             }
292
293             Token MakeLessThanToken()
294             {
295                 Position startPosition = position.Copy();
296                 TokenType ttype = TokenType.LESS_THAN;
297                 Advance();
298                 if (currentChar == '=')
299                 {
300                     ttype = TokenType.LESS_THAN_OR_EQUAL;
301                     Advance();
302                 }
303                 return new Token(ttype, startPosition, position);
304             }
305
306             Token MakeGreaterThanToken()
307             {
308                 Position startPosition = position.Copy();
309                 TokenType ttype = TokenType.GREATER_THAN;
310                 Advance();
311                 if (currentChar == '=')
312                 {
313                     ttype = TokenType.GREATER_THAN_OR_EQUAL;
314                     Advance();
315                 }
316                 return new Token(ttype, startPosition, position);
317             }
318         }
319     }
```

Listing 2: Token.cs

```
 1  namespace ShorkSharp
 2  {
 3      public class Token
 4      {
 5          public TokenType type { get; protected set; }
 6          public dynamic value { get; protected set; }
 7
 8          public Position startPosition { get; protected set; }
 9          public Position endPosition { get; protected set; }
10
11          public Token(TokenType type, Position startPosition)
12          {
13              this.type = type;
14              this.value = null;
15              this.startPosition = startPosition.Copy();
16              this.endPosition = startPosition.Copy();
17          }
18          public Token(TokenType type, Position startPosition, Position
                ↪ endPosition)
19          {
20              this.type = type;
21              this.value = null;
22              this.startPosition = startPosition.Copy();
23              this.endPosition = endPosition.Copy();
24          }
25          public Token(TokenType type, dynamic value, Position startPosition)
26          {
27              this.type = type;
28              this.value = value;
29              this.startPosition = startPosition.Copy();
30              this.endPosition = startPosition.Copy();
```

```
31          }
32          public Token(TokenType type, dynamic value, Position startPosition,
                ↪ Position endPosition)
33          {
34              this.type = type;
35              this.value = value;
36              this.startPosition = startPosition.Copy();
37              this.endPosition = endPosition.Copy();
38          }
39
40          public override string ToString()
41          {
42              if (value == null)
43                  return string.Format("[{0}]", type);
44              else
45                  return string.Format("[{0} : {1}]", type, value);
46          }
47      }
48  }
```

Listing 3: TokenType.cs

```
1   namespace ShorkSharp
2   {
3       public enum TokenType
4       {
5           NUMBER,
6           STRING,
7           BOOL,
8           NULL,
9
10          KEYWORD,
11          IDENTIFIER,
12
13          PLUS,
14          MINUS,
15          MULTIPLY,
16          DIVIDE,
17          EXPONENT,
18
19          EQUALS,
20          DOUBLE_EQUALS,
21          NOT_EQUALS,
22          LESS_THAN,
23          GREATER_THAN,
24          LESS_THAN_OR_EQUAL,
25          GREATER_THAN_OR_EQUAL,
26
27          DOT,
28          COMMA,
29
30          LPAREN,
31          RPAREN,
32          LBRACE,
33          RBRACE,
34          LBRACKET,
35          RBRACKET,
36
37          NEWLINE,
38          EOF
39      }
40  }
```