

P1 Parser 100% Complete

Julia Parser

Sharon Perry CPL 4308 Section 03

Ronak Patel
3-20-2022

Some of the work I performed was fixing my previous scanner. I had a block remover put in that would remove tokens of \t. This allowed me to parse my file with the correct size. I used recursive descent parsing to build the structure of my program and scanned for tokens using the grammar chart. The grammar chart helped enable me to figure out whether there was an error in my test cases. The parses write to the console about what actions they're taking building out the BNF of the test cases. Another thing I had to keep track of was what values were given from ID's which I kept in a list. Using the Visual Studio Debugger tool was how I was able to mark out where the index should be for all test cases and let me map out exactly where I wanted it incremented which was immensely helpful to avoid out of bound errors or logic errors. Environment.Exit(0) would notify the user that a certain aspect of the program failed and exited after encountering an error.

Files Used

Scanner.cs – scans for tokens, eliminates white space, and used TokenType to build a list of tokens.


TokenType.CS – an enum type that had all my types I needed to classify my tokens.

Parser.CS – allowed me to read off my list of scanned tokens and verify if they accurate Julia files. If so, then it creates a BNF form of my code that allows me to interpret it in my main file.

Program.cs – this main file held was returned to most of my lists.

Screenshots

TEST 1

 Microsoft Visual Studio Debug Console

```
(function, FUNCTION)
(a, IDENTIFIER)
((, LEFT_PARENTHESIS)
()), RIGHT_PARENTHESIS)
(x, IDENTIFIER)
(=, ASSIGNMENT)
(1, LITERAL_INT)
(print, PRINT)
((, LEFT_PARENTHESIS)
(x, IDENTIFIER)
()), RIGHT_PARENTHESIS)
(end, END)

<program> -> function id() <block> end
<block> -> <statement>
<statement> -> <assignment>
<assignment_statement> -> id <assignment operator> <arithmetic_expression>
<arithmetic_expression> -> <literal_integer>
<block> -> <statement>
<statement> -> <assignment>
<print_statement> -> print(<arithmetic_expression>)
<arithmetic_expression> -> <id>
<id> -> 1
end
```

TEST 2

Microsoft Visual Studio Debug Console

```
(function, FUNCTION)
(a, IDENTIFIER)
((, LEFT_PARENTHESIS)
(, RIGHT_PARENTHESIS)
(x, IDENTIFIER)
(=, ASSIGNMENT)
(1, LITERAL_INT)
(while, WHILE)
(<, LESSTHAN_OP)
(x, IDENTIFIER)
(4, LITERAL_INT)
(do, DO)
(x, IDENTIFIER)
(+, ADD_OP)
(=, ASSIGNMENT)
(x, IDENTIFIER)
(1, LITERAL_INT)
(end, END)
(print, PRINT)
((, LEFT_PARENTHESIS)
(x, IDENTIFIER)
(, RIGHT_PARENTHESIS)
(end, END)
```

```
<program> -> function id() <block> end
<block> -> <statement>
<statement> -> <assignment>
<assignment_statement> -> id <assignment operator> <arithmetic_expression>
<arithmetic_expression> -> <literal_integer>
<block> -> <statement>
<statement> -> <assignment>
<while_statement> -> while <boolean_expression> then <block> else <block> end
<arithmetic_expression> -> <id>
<id> -> 1
<arithmetic_expression> -> <literal_integer>
<block> -> <statement>
<statement> -> <assignment>
UNABLE TO PARSE STATEMENT
```

TEST 3

Microsoft Visual Studio Debug Console

```
(function, FUNCTION)
(a, IDENTIFIER)
((, LEFT_PARENTHESIS)
(, RIGHT_PARENTHESIS)
(x, IDENTIFIER)
(=, ASSIGNMENT)
(1, LITERAL_INT)
(if, IF)
(~, NULL)
(=, ASSIGNMENT)
(x, IDENTIFIER)
(1, LITERAL_INT)
(then, THEN)
(print, PRINT)
((, LEFT_PARENTHESIS)
(0, LITERAL_INT)
(, RIGHT_PARENTHESIS)
(else, ELSE)
(print, PRINT)
((, LEFT_PARENTHESIS)
(1, LITERAL_INT)
(, RIGHT_PARENTHESIS)
(end, END)
(end, END)
```

```
<program> -> function id() <block> end
```

```
<block> -> <statement>
```

```
<statement> -> <assignment>
```

```
<assignment_statement> -> id <assignment operator> <arithmetic_expression>
```

```
<arithmetic_expression> -> <literal_integer>
```

```
<block> -> <statement>
```

```
<statement> -> <assignment>
```

```
<if_statement> -> if <boolean_expression> then <block> else <block> end
```

```
IF STATEMENT ERROR
```