CS3361

Concepts of Programming Languages

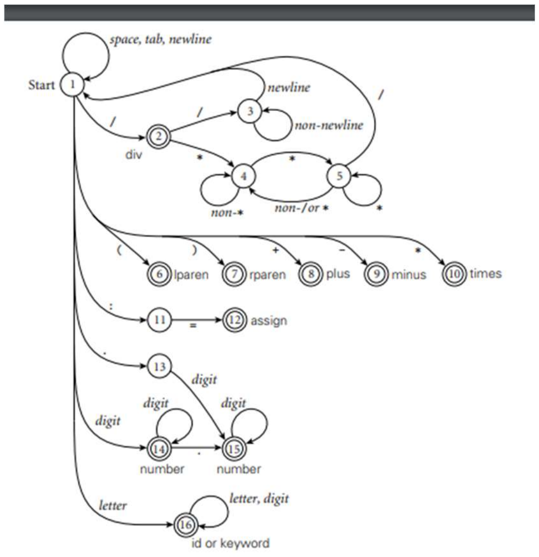
Team-based Project: Release # 2

Heath Byrd

Shashwath Udayakumar

Introduction

In this introduction, I will be detailing the structure of the code designed for parsing and building the xml tree. The parsing structure is directly designed after the following DFA:



Java methods are used to emulate the states of the automata machine, with different conditions resulting in the call of different states. By using the parser to grab a list of all tokens and token types from the user inputted text document, we can easily construct a matching xml tree. The use of ArrayLists to hold the list of characters from the txt file and the list of token and token types is allows you to easily add and retrieve the stored data across the program. Java methods allow you to call methods from within other methods and store data in the input data of the method, making a design that loops between all the methods in the manner pictured above feasible.

Pseudocode

Parser

```
public class Parser {

    //ArrayLists that will hold the list of tokens and their types that are pulled from the txt
document
```

```
static ArrayList<String> tokens = new ArrayList<>();
static ArrayList<String> tokenType = new ArrayList<>();

//List of characters from txt file and a list of keywords to be recognized by the parser
static ArrayList<Character> fileC = new ArrayList<>();
static ArrayList<String> keywords = new ArrayList<>();

public static void main(String[] args) {
  initialize();
}

public static void initialize() {
  addKeywords();

  //Asks the user to input the file path of the text document
  Scanner inputScanner = new Scanner(System.in);
  inputScanner.useDelimiter(System.lineSeparator());
  System.out.println("Please enter the path for the file you would like parsed:");
  String filePath = inputScanner.nextLine();

  try {
    scan(filePath);
  } catch (IOException ex) {
    Logger.getLogger(Parser.class.getName()).log(Level.SEVERE, null, ex);
  }
  //Begins the automata machine by entering the first state
  State1(0, 1);

  //Checks for tokens that returned as errors and upoin finding none, prints the xml tree
  if (tokens.contains("Error")) {
    System.out.println("\nError, your file contained an invalid token.");
  } else {
    xmlPrint();
  }
}

//Utility method to add standard keywords to arraylist for checking against
public static void addKeywords() {
  keywords.add("read");
  keywords.add("write");
}
```

XML print

```
public static void xmlPrint() {

    System.out.println("\n<Program>");
    System.out.println("\t<stmt_list>");
    System.out.println("\t\t<stmt>");

    for (int i = 0; i < tokens.size(); i++) {
      System.out.println("\t\t\t<" + tokenType.get(i) + ">");
      System.out.println("\t\t\t\t" + tokens.get(i));
      System.out.println("\t\t\t</" + tokenType.get(i) + ">");
    }

    System.out.println("\t\t</stmt>");
    System.out.println("\t</stmt_list>");
    System.out.println("</Program>");

  }

  //This method takes the entirety of the txt file and puts it into an arraylist of characters
  public static void scan(String fileName) throws FileNotFoundException, IOException {

    BufferedReader bufReader = new BufferedReader(new FileReader(fileName));

    String line = bufReader.readLine();
    while (line != null) {
      for (int i = 0; i < line.length(); i++) {
        fileC.add(line.charAt(i));
      }
      fileC.add('\n');
      line = bufReader.readLine();
    }
    bufReader.close();
  }

  //Utility method to pull the token strings from the arraylist
  public static String pullString(int index, int length) {
    String s = "";
    for (int i = index - length + 1; i <= index; i++) {
      s += fileC.get(i);
    }
```

```
        return s;
    }
```

DFA states

```
    //All methods titled State(Number) are designed to model a deterministic finite automata for
the purpose of parsing tokens,
    //the states are interconnected and lead to each other in the exact same manner as a standard
DFA
    public static void State1(int index, int length) {
        if (index >= fileC.size()) {
        } else if (fileC.get(index).equals(' ') || fileC.get(index).equals('\t') ||
fileC.get(index).equals('\n')) {
            State1(index + 1, 1);
        } else if (fileC.get(index).equals('/')) {
            State2(index + 1, length + 1);
        } else if (fileC.get(index).equals('(')) {
            tokens.add(pullString(index, length));
            tokenType.add("lparen");
            State1(index + 1, 1);
        } else if (fileC.get(index).equals(')')) {
            tokens.add(pullString(index, length));
            tokenType.add("rparen");
            State1(index + 1, 1);
        } else if (fileC.get(index).equals('+')) {
            tokens.add(pullString(index, length));
            tokenType.add("plus");
            State1(index + 1, 1);
        } else if (fileC.get(index).equals('-')) {
            tokens.add(pullString(index, length));
            tokenType.add("minus");
            State1(index + 1, 1);
        } else if (fileC.get(index).equals('*')) {
            tokens.add(pullString(index, length));
            tokenType.add("times");
            State1(index + 1, 1);
        } else if (fileC.get(index).equals(':')) {
            State11(index + 1, length + 1);
        } else if (fileC.get(index).equals('.')) {
            State13(index + 1, length + 1);
        } else if (Character.isDigit(fileC.get(index))) {
            State14(index + 1, length + 1);
        } else if (Character.isAlphabetic(fileC.get(index))) {
            State16(index + 1, length + 1);
        }

    }

    public static void State2(int index, int length) {
        if (index >= fileC.size()) {

        } else if (fileC.get(index).equals('/')) {
            State3(index + 1, length + 1);
        } else if (fileC.get(index).equals('*')) {
            State4(index + 1, length + 1);
        } else if (fileC.get(index).equals(' ') || fileC.get(index).equals('\t') ||
fileC.get(index).equals('\n') || index == fileC.size()) {
            tokens.add(pullString(index, length - 1));
            tokenType.add("div");
            State1(index + 1, 1);
        } else {
            tokens.add("Error");
            tokenType.add("Error");
            State1(index + 1, 1);
        }
    }

    public static void State3(int index, int length) {
        if (index >= fileC.size()) {

        } else if (fileC.get(index).equals('\n')) {
            State3(index + 1, length + 1);
        } else {
            State1(index + 1, 1);
        }
    }

    public static void State4(int index, int length) {
        if (index >= fileC.size()) {

        } else if (fileC.get(index).equals('*')) {
            State5(index + 1, length + 1);
        } else {
            State4(index + 1, length + 1);
        }
    }

    public static void State5(int index, int length) {
        if (index >= fileC.size()) {
        } else if (fileC.get(index).equals('*')) {
            State5(index + 1, length + 1);
        } else if (fileC.get(index).equals('/')) {
            State1(index + 1, 1);
        } else {
            State4(index + 1, length + 1);
        }
    }

    public static void State11(int index, int length) {
        if (index >= fileC.size()) {

        } else if (fileC.get(index).equals('=')) {
            tokens.add(pullString(index, length));
            tokenType.add("assign");
            State1(index + 1, 1);
        } else {
            tokens.add("Error");
            tokenType.add("Error");
            State1(index + 1, 1);
        }
    }

    public static void State13(int index, int length) {
        if (index >= fileC.size()) {

        } else if (Character.isDigit(fileC.get(index))) {
            State15(index + 1, length + 1);
        } else {
            tokens.add("Error");
            tokenType.add("Error");
            State1(index + 1, 1);
        }
    }

    public static void State14(int index, int length) {
        if (index >= fileC.size()) {

        } else if (Character.isDigit(fileC.get(index))) {
            State14(index + 1, length + 1);
        } else if (fileC.get(index).equals(' ') || fileC.get(index).equals('\t') ||
fileC.get(index).equals('\n')) {
            tokens.add(pullString(index - 1, length - 1));
            tokenType.add("number");
            State1(index + 1, 1);
        } else if (fileC.get(index).equals('.')) {
            State15(index + 1, length + 1);
        } else {
            tokens.add("Error");
            tokenType.add("Error");
            State1(index + 1, 1);
        }
    }

    public static void State15(int index, int length) {
        if (index >= fileC.size()) {

        } else if (Character.isDigit(fileC.get(index))) {
            State15(index + 1, length + 1);
        } else if (fileC.get(index).equals(' ') || fileC.get(index).equals('\t') ||
fileC.get(index).equals('\n') || index == fileC.size() - 1) {
            tokens.add(pullString(index - 1, length - 1));
            tokenType.add("number");
            State1(index + 1, 1);
        } else {
            tokens.add("Error");
            tokenType.add("Error");
            State1(index + 1, 1);
        }
    }

    public static void State16(int index, int length) {
        if (index >= fileC.size()) {
            String test = pullString(index - 1, length - 1);
            if (keywords.contains(test)) {
                tokens.add(test);
                tokenType.add(test);
                State1(index + 1, 1);
            } else {
                tokens.add(test);
                tokenType.add("id");
                State1(index + 1, 1);
            }
        } else if (Character.isDigit(fileC.get(index)) || Character.isAlphabetic(fileC.get(index))) {
            State16(index + 1, length + 1);
        } else if (fileC.get(index).equals(' ') || fileC.get(index).equals('\t') ||
fileC.get(index).equals('\n')) {
            String test = pullString(index - 1, length - 1);
            if (keywords.contains(test)) {
                tokens.add(test);
                tokenType.add(test);
                State1(index + 1, 1);
            } else {
                tokens.add(test);
                tokenType.add("id");
                State1(index + 1, 1);
            }
        } else {
            tokens.add("Error");
            tokenType.add("Error");
            State1(index + 1, 1);
        }
    }
```

Test cases

Test 1
Input File:

```
A
+
read
```

Output:

```
<Program>
	<stmt_list>
		<stmt>
			<id>
				A
			</id>
			<plus>
				+
			</plus>
			<read>
				read
			</read>
		</stmt>
	</stmt_list>
</Program>
```

Test 2
Input File:

```
How are you today?
```

Output:

```
Error, your file contained an invalid token.
```

Test 3
Input File:

```
(
write
why
.02
```

Output:

```
<Program>
	<stmt_list>
		<stmt>
			<lparen>
				(
			</lparen>
			<write>
				write
			</write>
			<id>
				why
			</id>
			<number>
				.02
			</number>
		</stmt>
	</stmt_list>
</Program>
```

Test 4
Input File:

```
A sunny day 37
78
//This is a comment
```

Output:

```
<Program>
```

```
		<stmt_list>
			<stmt>
				<id>
					A
				</id>
				<id>
					sunny
				</id>
				<id>
					day
				</id>
				<number>
					37
				</number>
				<number>
					78
				</number>
			</stmt>
		</stmt_list>
	</Program>
```

Acknowledgement

Heath Byrd

Shashwath Udayakumar