Comp-2140: Lab Assignment 5–Recursive Descent Parser (8 Marks)

Useful links

- Submission site
- Tiny Language Grammar and links for JavaCup and JLex
- Step-by-step instructions

1 Submission and Due date

You should submit your code at the A5 Submission Site (you can click the left highlighted words to go to the submission site).

2 Purpose of the Assignment

The purpose of this assignment is to understand that topdown recursive parser is easy to construct manually, but it is inefficient when it runs. We learn how to parse a language using recursive descent method; learn how to use a scanner. We have learnt how to generate and use a scanner in A4. In A4, scanner, parser, and their generators are all mixed. Now you need to identify the pertinent classes for the scanner, and learn how to invoke the scanner.

3 Assignment specification

You are required to write a Java program that parses programs written in our Tiny language, without the help of a parser generator. Please follow the recursive-decent parser example given in the lecture. You only need to judge whether a Tiny program is syntactically correct or not. You are not required to handle the ambiguity of the grammar.

You will write a Java class called A5.java, which is the parser for our Tiny language. The main method in A5.java should be:

```
public static void main(String[] args) throws Exception {
        //construct the token array
        BufferedWriter bw=new BufferedWriter(new FileWriter("a5.output"));
        A5Scanner scanner = new A5Scanner(new FileInputStream(new File("A5.tiny")));
        // note that yylex() is the default method to get the next token in scanner that is
            generated by JLex.
        Symbol token;
        while ((token=scanner.yylex()).sym!= A5Sym.EOF) {
                tokens.add(token);
        }w
        tokens.add(token);    // add EOF as the last token in the array
        boolean legal= program() && nextToken().sym==A5Sym.EOF;
        bw.write((legal)?"legal":"illegal");
        bw.close();
}
```

The commands to test your program are:

```
javac A5.java A5Scanner.java A5Sym.java Symbol.java
java A5
```

Please make sure that those two commands can run without any other classes. That is, to test your program, you should create a clean directory with no other classes, no JLex, no JavaCup. All you have in this directory are those 4 classes and the input file.

If a5.input is a correct Tiny program, the output file a5.output will contain a single word "legal"; otherwise a word "illegal". Please note that the program may run several minutes to process our sample Tiny program. Notice how slow this program is and why it is better to use other parsing methods.

In addition to the parser class itself, you need to define the following auxiliary classes:

- The scanner called A5Scanner.java, which is generated using JLex. You can reuse the scanner that is produced in previous assignments. The name of the scanner is called A5Scanner. Note the function to get next token is called yylex().
- Symbol.java: this is the class to represent the tokens. You should use the Symbol.java class from *java_cup.runtime* package.
- A5Sym.java: this is the class to store the types of the tokens. You can either manually produce that class, or reuse the token type class that is generated in assignment 4.

4 What to submit

You need to submit the following java files:

- Your java parser named A5.java
- Your scanner named A5Scanner.java
- Your symbol type class named A5Sym.java

You do not need to submit Symbol.java. It is the same for everyone, and we will provide that class when we run the marking program.

5 Marking scheme

```
yourMark=0;
for (each of the 10 tests A5.tiny) {
    if (your program runs longer than 5 minutes) {
        break;
    }
    if (A5.tiny is legal program && a5.output says "legal")
          yourMark+=0.8;
     if (A5.tiny is not a correct Tiny program && a5.output says "illegal" )
          yourMark+=0.8;

 }
```

6 Bonus Mark: Rewrite the Grammar into LL(1)

The formula for the bonus mark is

```
 bonus_mark=(running_time<1.5s)? 0.5*1.5/running_time:0
```

To improve the speed, you need to left factor the grammar, and ideally to turn the grammar into LL(1). I will also look at your code to verify that your code is fast because of the grammar rewriting, not that you used LALR parsing.

7 Academic Integrity

By submitting on the web site, you verify that the submitted work is your own and adheres to all my Academic Rights and Responsibilities as outlined in the Student Code of Conduct.