CITY
UNIVERSITY OF LONDON
EST 1894

IN2009: Language Processors
Coursework 1: Parsing

Introduction

This coursework requires you to use JavaCC to implement a complete parser for the LPL language. The LPL grammar is available in Moodle. Implementing a parser for some other grammar, or implementing a parser without using JavaCC, will gain no marks.

Key Information

Note: *Test before you submit*. Assessment will be based on automated tests. Submissions which do not compile with JavaCC, or for which JavaCC generates Java code which does not compile, will therefore get zero marks.

Module marks
This coursework is worth 30% of the coursework marks for the module. This coursework is marked out of 100.

Submissions
You must submit your solution in Moodle as a single zip file containing your `lpl` source folder complete with the relevant sub-folders and files. The relevant contents are listed at the end of this document; it is permissible to include additional files in your submission but they will be discarded. **Note**: do not use rar or any other archive format; it must be a zip file.

Marks & Feedback
Marks and feedback will be available no later than 3 weeks after the submission deadline.

Pair-working
Pair-working is permitted only if you officially registered as a pair at the start of term. If working as a pair, both members must submit identical files and both members must contribute equally to the work.

Plagiarism
If you copy the work of others (either that of fellow students or of a third party) with or without their permission, you will score no marks and disciplinary action for Academic Misconduct will follow.

The Task

Download `LPL.zip` and extract the files. This provides a folder of test inputs, a compiled LPL pretty-printer (`LPLpp.jar` - more about this below) and the source folder `lpl`, which contains a number of files and sub-folders, including:

- `lpl/JustLex.java:` you can use this to test your token definitions.
- `lpl/Parse.java:` you can use this to test whether your parser recognises valid LPL syntax.
- `lpl/Print.java`: you can use this to test if your parser is building the correct AST structure (see below for more details).
- `lpl/parser/LPL.jj`: a partially implemented LPL parser which you must complete.
- `lpl/ast`: the AST classes. These are fully implemented except for their `toString()` methods, which you must add (some are done for you).

The required workflow should be familiar from the class exercises (if you have not yet completed the class exercises you are strongly advised to do so before you embark on the coursework).

1. [50%] Complete the parser in `LPL.jj` so that it recognises all (and only) syntactically valid LPL programs . **Notes**:
    - While working on this part, you can simply return null in your semantic actions.
    - To just test your token definitions, separately from your parsing logic, use `lpl.JustLex` instead of `lpl.Parse`.
    - The grammar avoids ambiguity, but, as discussed in the lectures, there are still left-recursion and lookahead issues that you will need to resolve.
    - To test your parser, run `lpl.Parse`, providing a test-input file name on the command line, for example:

    ```
    java lpl.Parse test-inputs/simple/add.lpl
    ```
2. [50%] Add semantic actions to your parser to build the AST, and implement `toString()` methods in all of the AST classes (some have been done for you). **Notes**:

    - There is no requirement for your `toString()` methods to provide nicely formatted output. However, the ouput produced by your `toString()` methods must be syntactically valid LPL (it should be accepted by any correct LPL parser).

    - To test this part, use `lpl.Print`. For example:

    ```
    java lpl.Print test-inputs/simple/add.lpl
    ```

    The output should agree with the input. However, since the output formatting (spaces and newlines) will be different from the input, it may be difficult to check for correctness. To help with this, first run a test as above to make sure that you don’t get any parse exceptions or lexical errors (if you do, there is something wrong with your parser). Then you can use the LPL pretty printer that was included in the zip file, as follows (all on one line):

    ```
    java lpl.Print test-inputs/simple/add.lpl |
                              java -cp LPLpp.jar lpl.PrettyPrint
    ```

This command pipes the output from your print test into the pretty printer, which then parses it as an LPL program and re-outputs it nicely formatted. The result should be the same as the test input (except for any comments, which will not be preserved). If you get a parse exception or lexical error, or if the output is different from the test input, you either have an error in your `toString()` methods, or your parser is building the wrong AST, or both.

Zip file submission: required contents

```
Archive:  coursework1-submission.zip
Name
----
lpl/
lpl/parser/
lpl/parser/LPL.jj
lpl/ast/
lpl/ast/Exp.java
lpl/ast/ExpArrayLength.java
lpl/ast/ExpArrayLookup.java
lpl/ast/ExpCall.java
lpl/ast/ExpOp.java
lpl/ast/ExpPrimaryExp.java
lpl/ast/Formal.java
lpl/ast/FunDef.java
lpl/ast/PrimaryExp.java
lpl/ast/PrimaryExpExp.java
lpl/ast/PrimaryExpFalse.java
lpl/ast/PrimaryExpInteger.java
lpl/ast/PrimaryExpIsnull.java
lpl/ast/PrimaryExpNewArray.java
lpl/ast/PrimaryExpNot.java
lpl/ast/PrimaryExpTrue.java
lpl/ast/PrimaryExpVar.java
lpl/ast/Program.java
lpl/ast/Stm.java
lpl/ast/StmArrayAssign.java
lpl/ast/StmAssign.java
lpl/ast/StmBlock.java
lpl/ast/StmCall.java
lpl/ast/StmIf.java
lpl/ast/StmOutchar.java
lpl/ast/StmOutput.java
lpl/ast/StmReturn.java
lpl/ast/StmWhile.java
lpl/ast/Type.java
lpl/ast/TypeArray.java
lpl/ast/TypeBoolean.java
lpl/ast/TypeInt.java
lpl/ast/TypeUnit.java
lpl/ast/VarDecl.java
-------
38 files (file count includes folders and sub-folders)
```