1 Recursive Descent – LL(1) Grammar

The grammar given on INGInious needs to be modified in order to not be left-recursive. The following presents an equivalent grammar which is LL(1) (its first-sets and follow-sets, as well as the parsing table are given in the code file):

E ::= TE'

E' ::= or TE' | ε

T ::= FT'

T' ::= and FT' | ε

F ::= not F | (E) | id

2 Programming Directly in Java Bytecode

Using `CLEmitter`, it is quite simple to create a "hand-written" class file. In order to specify the output directory for the generated file, one can use the `destinationDir` method. A `CLEmitter` uses an `ArrayList` to store the various instructions that are needed. Similarly to how this was done in the provided examples of `CLEmitter` usage on the lecture slides and in the `tests/clemitter` folder, the class is created using the `addClass` method, and an implicit no-arg constructor is added. For the `gcd` method, `ClassToGenerate` mainly uses `iload_0`, `iload_1` (to load the first and second arguments, resp.), `istore_0`, `istore_1` (to (over)write the first and second arguments, resp.), as well as the `isub` command to subtract two numbers. These are all called using `CLEmitters`'s `addNoArgInstruction` method.

In order to implement the various control structures (the `while` loop and `if` block), `CLEmitter` has an `addBranchInstruction` method, which takes a first argument with the branch instruction to execute (`goto` or `if_icmple`), and a second argument with the name of the label to which one needs to branch. Labels can be specified using `CLEmitter`'s `addLabel` instruction, which takes a `String` as argument.

3 Lexical Analysis

3.1 Hand-written Compiler

In the `Scanner`, skipping over block comments is done by first detecting the start of a block comment in the `getNextToken` method. Once this has been detected, one enters a `while` loop which exits as soon as the end of the block comment has been detected.

3.2 JavaCC Compiler

Using JavaCC and lexical states, a neat solution is the following:

```
 SKIP: {"/*":  IN_BLOCK_COMMENT}
< IN_BLOCK_COMMENT > SKIP: {< ~[] >}
< IN_BLOCK_COMMENT > SKIP: {"*/":  DEFAULT}
```

It works as follows: when the start of a block comment is matched in the `DEFAULT` state, it is skipped and the state is switched to `IN_BLOCK_COMMENT`. Any other character seen in this new state is skipped, unless JavaCC sees the end of a block comment, in which case this is skipped and the state is switched back to `DEFAULT`.

4 Parsing and Semantic Analysis

4.1 Hand-written Compiler

4.1.1 Conditional-or (`||`)

In order to implement the conditional-or expression in the hand-written compiler, a few tweaks needed to be made. Logical precedence entails that the conditional-or expression would have lower priority than the conditional-and expression which already existed. This meant that any lower priority expression using conditional-and expressions would now have to use conditional-or expressions, which in turn are made up of conditional-and expressions separated by "`||`". The implementation for this expression was very similar to the one for the conditional-and expression, once this priority-related issue was resolved. Changes were made in

Gilles Peiffer (24321600), Liliya Semerikova (64811600)

- the `TokenKind` `enum`, to which one needed to add a new kind of token (`LOR("||")`);
- the `Scanner` class (where one simply needed to add a possibility to the `getNextToken` method);
- the `Parser` class (where a new `conditionalOrExpression` method was added);
- the `JBooleanBinaryExpression` class was extended with a class for logical-or operations, `JLogicalOrOp`.

4.1.2 Do-while (`do {} while ();`)

Adding the `do while` control structure was made easier thanks to its similarity with the existing `while` structure. The only difference between the two is that the body of the former is always executed at least once, whereas for the latter this is not the case. The implementation thus consisted in changing the following:

- the `TokenKind` `enum`, to which one needed to add a new kind of token (`DO("do")`);
- the `Scanner` class (to define the new `do` keyword);
- the `Parser` class (where another possibility was added to the `statement` method);
- the `JDoWhileStatement` was added, which is based on the `JWhileStatement` class but reflects their structural differences.

4.2 JavaCC Compiler

Fundamentally, the changes to implement these operations on the JavaCC compiler were exactly the same as the ones for the hand-written compiler, except for the fact they are all localized in the `j--.jj` file.

4.2.1 Conditional-or (`||`)

As before, a new operator (`< LOR: "||" >`) was defined, and a new function `conditionalOrExpression` was added to represent conditional-or expressions (and their precedence with respect to other expressions).

4.2.2 Do-while (`do {} while ();`)

To implement the `do while` statement with JavaCC, `do` was declared as a reserved word, and a new possibility was added to the `statement` function for the `do while` statement.

5 Tests

To test the correctness of the `Generator` class, one can use the Fernflower (or any other) decompiler, which translates a compiled class file into a regular java file. This output can then be compared with the original java file. The `javap` command provides a similar (though more rudimental) functionality.

In order to test whether the new language constructs work, tests were added in a similar manner to the one explained in Chapter 1.5 of [1], for each functionality: “pass” tests allowing one to check whether execution behaves as expected, and “fail” tests to verify compilation stops graciously in case of errors. Test files are provided in `pass/MultilineComment.java`, `pass/Or.java`, `pass/DoWhile.java`, `junit/OrTest.java`, `junit/DoWhile.java`, `fail/OrErrors.java`, `fail/DoWhileErrors.java`, and the test suite provided in `junit/JMinusMinusTestRunner.java` was extended accordingly.

6 Updated Grammar

The changes to the `j--` grammar can be summarized as follows:

- Block comments were added, which can be represented by the following JavaCC-style regex: “`"/*" (~["*"])* "*" (~["*","/"] (~["*"])* "*" | "*")* "/"`”.
- The conditional-or expression was added, which has precedence level 11 and can be represented as “`conditionalOrExpression ::= conditionalAndExpression LOR conditionalAndExpression`”.
- The `do while` statement was added, which can be represented as “`DO statement WHILE parExpression SEMI`”.

References

[1] Bill Campbell, Swami Iyer, and Bahar Akbal-Delibaş. *Introduction to Compiler Construction in a Java World*. Chapman and Hall/CRC, Boca Raton, Florida, 2012.