Java Parser Project

Irfan Sheikh, Vineet Kumar

June 2021

Contents

1 Introduction

The objective of this project is to parse a Java For-Each loop (also referred to as the Enhanced For loop). We start with the Lexical Analysis, using Lex, which returns a sequence of tokens corresponding to lexemes in the input file. The parser then tries to derive this sequence of tokens from the rules of the grammar that we list in the yacc specification file. If successful, then the program is syntactically valid. Otherwise, there exist syntax errors in the input program.

2 The Basic Java Language

2.1 Lexical Analysis

The tokens returned by the lexical analyzer are:
`REAL, INTEGER` : floating point and integer numbers respectively.
`BASIC, SEQUENCE_TYPE` : basic types like `int, char`, etc, and sequence types like `ArrayList`.
`ID` : identifiers

For more details regarding the regular expressions of these tokens, the Lex file `src/basic.l` can be referred, which contains comments that also explain the handling of reserved keywords. Any other characters are returned as is. As per the grammar, other characters that are part of valid syntax are
`( ) ; : = + - * / { } < > [ ]`

2.2 Syntax Analysis

The grammar for a subset of the Java language is as follows.

```
program → block
block → { decls stmts }

decls → decls decl | ϵ
decl → type ID assignment ;

type → sequence_type | array_type | basic
basic → BASIC
array_type → BASIC [ ]
sequence_type → SEQUENCE_TYPE < BASIC >

assignment → = expr
       | = NEW BASIC [ INTEGER ]
       | = NEW sequence_type ( )
       | ϵ

stmts → stmts stmt | ϵ
stmt → ID assignment ; | ID [ INTEGER ] = expr ;
       | foreach stmt | { stmts }

foreach → FOR ( BASIC ID : ID )

expr → expr + term
       | expr - term
       | term
```

```
term → term * factor
       | term / factor
       | term

factor → ( expr ) | ID | INTEGER | REAL
```

For more details regarding the grammar, the Yacc specification file `src/basic.y` can be referred, which contains comments explaining important parts of the grammar.

3 Assumptions

We have assumed the following points:

1. The grammar only allows programs that have statements followed by declarations, enclosed within curly braces.
2. The declarations have an optional assignment component.
3. The statements can only be either assignment statements, or a for-each construct, inside of which there can be one or more statements.
4. In a for-each loop, the underlying type of the sequence (or array) can be of only a primitive data type.
5. No Object Oriented Programming aspects of the Java Language have been taken into consideration.

4 Features

In addition to the lexical and syntax analysis, we have implemented the following features:

1. Symbol table is being maintained. For more details, the files `include/symtab.h` and `src/symtab.c` can be referred.
2. Using the symbol table, re-declaration of identifiers is being checked during the parsing process. Additionally, we also check whether or not an identifier being used in a statement has been previously declared.
3. Basic type checking is being performed whenever an identifier is being assigned something.
4. When parsing a for-each loop, in addition to (2), we check that the identifier being iterated is indeed a sequence, and, we check whether or not the underlying type of the sequence is the same as that of the looping variable.

5 Test Cases

All test cases reside in the `tests` directory. The bash script in the root directory of the project, `run.sh`, provides a simple way to run these tests. For example, `./run.sh tests/basic.java` will compile the lexical analyzer, the parser, and run the executable with the file `tests/basic.java`. The commands inside the bash script can also be executed manually, if one wishes to. The program will print whether or not the input program `program.java` is syntactically valid. If it is, then the file `program.java.txt` is created, which lists any re-declaration errors, type errors, and, the symbol table.

5.1 First Test Case : Valid For-Each Loop

5.1.1 Program Code

The file `tests/foreach1.java` contains a simple program with a **valid** for-each loop.

```
{
    int sum = 0;
    int[] arr = new int[10];

    for (int elem : arr) {
        sum = sum + elem;
    }
}
```

5.1.2 Output

```

Syntactically valid program!
Generated output containing symtab and type errors (if any) in
    tests/foreach1.java.txt

Contents of tests/foreach1.java.txt:

foreach symbols: int, elem, arr
Syntactically valid and type compatible foreach loop

_____________________________________________________
no. name    type    token   is_sequence
_____________________________________________________
#41 sum int 258 0
#42 arr int 258 1
#43 elem    int 258 0

_____________________________________________________
```

5.2 Second Test Case : Invalid For-Each Loop

5.2.1 Program Code

The file `tests/foreach2.java` contains a simple program with an **invalid** for-each loop.

```
{
    int sum = 0;
    int[] arr = new int[10];

    for int elem : arr {
        sum = sum + elem;
    }
}
```

5.2.2 Output

```

error: syntax error
error: Invalid Syntax!

error: Invalid Syntax!
...
```

5.3 Third Test Case : Checking Type Errors

The file `tests/basic.java` contains a syntactically valid program, but with a lot of re-declaration and type errors, as noted in the comments.

5.3.1 Program Code

```
{









    int a; int b;
    char c; boolean d;

    // should cause incompatible types
    int e = 10.0;
    double j;

    char[] str = new char[100];

    ArrayList<int> arr = new ArrayList<int>();
    int[] x = new int[10];
```

```
25      int[] hello = new int[100];
26
27      ArrayList<int> arr1;
28      arr1 = new ArrayList<int>();
29
30      // valid foreach loop
31      for (int elem : arr) {
32          a = a * b + elem;
33      }
34
35      // should cause d is not a sequence
36      for (int g : d) {
37          a = a + 1;
38      }
39
40
41      x[0] = 12;
42      x[1] = 13;
43      x[2] = 16;
44
45      // should cause incompatible types int and double
46      x[3] = 19.87;
47
48      e = 10;
49
50      // incompatible types boolean and int
51      d = 10 + 20 * 30;
52
53      // should cause f is not declared
54      a = b + e + f;
55
56      // should cause incompatible types double and int
57      j = 10;
58
59      j = 10.23; // ok
60
61      // should cause redeclaration of x
62      for (int x : arr) a = a + 1;
63
64      // should cause "a" is not a sequence"
65      for (int element : a) {
66          a = a + 1;
67      }
68
69      // should cause incompatible types int and char
70      for (int ele : str) {
71          ele = ele + 1;
72      }
73
74  }
```

5.3.2 Output

```
1
2  Syntactically valid program!
3  Generated output containing symtab and type errors (if any) in
       tests/basic.java.txt
4
5  Contents of tests/basic.java.txt:
6
7  ERROR (declaration) : Incompatible types e (int) and (double)
8
9
10 foreach symbols: int, elem, arr
11 Syntactically valid and type compatible foreach loop
12
13
14 foreach symbols: int, g, d
15 ERROR (foreach): Symbol d is not a sequence.
16
17 ERROR (array assignment): Incompatible types x (int) and expr (
       double)
18
19 ERROR (assignment): Incompatible types d (boolean) and int
20 ERROR (factor -> ID): Symbol f is not declared!
21
22 ERROR (assignment): Incompatible types j (double) and int
23
24
25 foreach symbols: int, x, arr
26 ERROR (foreach): Redeclaration of symbol x
27
28
29 foreach symbols: int, element, a
30 ERROR (foreach): Symbol a is not a sequence.
31
32
33 foreach symbols: int, ele, str
34 ERROR (foreach): Incompatible types: str (char) and  ele (int)
35
36 ____________________________________________________
37 no. name    type    token   is_sequence
38 ____________________________________________________
39 #41 a   int 258 0
40 #42 b   int 258 0
41 #43 c   char    258 0
42 #44 d   boolean 258 0
43 #45 e   int 258 0
```

```
#46 j   double  258 0
#47 str char    258 1
#48 arr int 258 1
#49 x   int 258 1
#50 hello   int 258 1
#51 arr1    int 258 1
#52 elem    int 258 0
#53 g   int 258 0
#54 element int 258 0
#55 ele int 258 0

_____________________________________________________
```