PROJECT ABSTRACT

CS6109 - COMPILER DESIGN

Parser for basic java programs using LEX and YACC

Kariketi Tharun Reddy, 2018103034

Vivek Ramkumar, 2018103082

G. R. Srikanth, 2018103603

Objective:

To write a compiler that converts a high-level language, namely Java, into machine code by following the core principles of compiler design. Compiler construction, in its entirety, takes into account thousands upon thousands of patterns from a source language, in order to achieve high flexibility in programming. We provide a miniature version, which understands a limited set of instructions, but nonetheless just as powerful.

Introduction:

The general structure of a compiler consists of:

Analysis: Lexical, syntax and semantic analyzers

Synthesis: Code generation and optimization

We will be writing a lexical analyzer in the Lex language, and the syntax as well as the semantic analyzers in the Yacc language. The code will be generated and cleaned up by using shell scripts.

The output - an intermediate code representation - will be generated and written into a text file. We will use a three-address code to run-compile the program separately, for evaluation purposes. The result can be seen in the command line itself.

Functionalities:

This project consists of the basic functions for **number manipulation and boolean comparisons** and can perform **complex calculations with multiple parentheses.** It can analyze a script written in Java, consisting of the basic number functions, and generate intermediate code, which will then be optimized. The print statement is also understood by the compiler's analysis tools, which can be used to obtain the result visually.

Main arithmetic operations:

The main arithmetic operations available are +, -, *, /, %.

Conditional statements:

This programming language supports various conditional operations like <, >, ==. <=, >=. It also supports assigning direct boolean values in the form of keywords 'True' and 'False'.

Variables:

The variables can be assigned from [a-z] and [A-Z]. The symbol table implementation involves a simple symbol table of 52 rows. And a simple hashing algorithm to fill in according to the ascending value of the variables is used. The result of a conditional statement can also be stored inside a variable.

Printing the result:

The results can be print using the 'print' statement followed by an expression or an identifier.

Sample program:

```
Int a=10+5/5+5+2*1 ;
Int b = 6%2;
Int c = a+b;
System.out.println(c);
System.out.println(500);
```

Output (Evaluation):

```
18
500
```

Output (Three address code):

```
k0 := 5 / 5
k1 := 10 + k0
k2 := k1 + 5
k3 := 2 * 1
k4 := k2 + k3
a := k4
k5 := 6 % 2
b :=k5
k9 := a + b
c := k9
print c
print 500
```

Conclusion:

The compiler in regard will be constructed by adhering to aforementioned rules, and by following the proven rules of compiler design. It is purely experimental, and is intended to provide an insight into how compilers work from the inside-out. It must be noted that the abstract here is not a representation of the final compiler. It can be extended at any time during the creation of the analysis and synthesis components. This project was carried out as part of the CS6109 course in compiler design, as prescribed.