## ASSIGNMENT B4

**TITLE :YACC program to validate variable declarations**

**Problem Statement:**
Write a program using YACC specifications to implement syntax analysis phase of the compiler to validate type and syntax of variable declaration in Java.

**Objectives:**

- Be proficient on writing grammars to specify syntax

- Understand the theories behind different parsing strategies-their strengths and limitations

- Understand how the generation of parser can be automated

- Be able to use YACC to generate parsers

**Outcomes:**
I will be able to understand and implement YACC specifications for syntax analysis phase of a compiler.

**Software and Hardware Requirements:**

- Working PC.

- 64 bit Fedora OS

- Eclipse IDE and JAVA

- I3 processor

- LEX and YACC

**Theory**
Lex recognizes regular expressions, whereas YACC recognizes entire grammar. Lex divides the input stream into tokens, while YACC uses these tokens and groups them together logically.

YACC stands for Yet Another Compiler Compiler. YACC provides a tool to produce a parser for a given grammar. YACC is a program designed to compile a LALR (1) grammar. It is used to produce the source code of the syntactic analyzer of the language produced by LALR (1) grammar.
**The syntax of a YACC file:**
% {
**declaration section**
% }
**rules section**
% %

**user defined functions**

### Declaration section:

Here, the definition section is the same as that of Lex, where we can define all tokens and include header files. The declarations section is used to define the symbols used to define the target language and their relationship with each other. In particular, much of the additional information required to resolve ambiguities in the context-free grammar for the target language is provided here.

### Grammar Rules in Yacc:

The rules section defines the context-free grammar to be accepted by the function Yacc generates, and associates with those rules C-language actions and additional precedence information. The grammar is described below, and a formal definition follows.

The rules section consists of one or more grammar rules. A grammar rule has the form:

A : BODY ;

The symbol A represents a non-terminal name, and BODY represents a sequence of zero or more names, literals, and semantic actions that can then be followed by optional precedence rules. Only the names and literals participate in the formation of the grammar; the semantic actions and precedence rules are used in other ways. The colon and the semicolon are Yacc punctuation.

If there are several successive grammar rules with the same left-hand side, the vertical bar '| ' can be used to avoid rewriting the left-hand side; in this case the semicolon appears only after the last rule. The BODY part can be empty.

### Programs Section

The programs section can include the definition of the lexical analyzer yylex(), and any other functions; for example, those used in the actions specified in the grammar rules. It is unspecified whether the programs section precedes or follows the semantic actions in the output file; therefore, if the application contains any macro definitions and declarations intended to apply to the code in the semantic actions, it shall place them within "% { ... % } " in the declarations section.

### Interface to the Lexical Analyzer

The yylex() function is an integer-valued function that returns a token number representing the kind of token read. If there is a value associated with the token returned by yylex() (see the discussion of tag above), it shall be assigned to the external variable yylval.

### TEST CASES:

| INPUT | OUTPUT (Expected) | OUTPUT (Actual) | RESULT |
|---|---|---|---|
| int a= 10; | int => TYPE<br>a => IDENTIFIER<br>= => ASSIGN<br>10 => NUMBER<br>; =>SC<br>VALID VARIABLE DECLARATION | int => TYPE<br>a => IDENTIFIER<br>= => ASSIGN<br>10 => NUMBER<br>; =>SC<br>VALID VARIABLE DECLARATION | Success |
| int b=5; | int => TYPE<br>b => IDENTIFIER<br>= => ASSIGN<br>5 => NUMBER<br>; =>SC<br>VALID VARIABLE DECLARATION | int => TYPE<br>b => IDENTIFIER<br>= => ASSIGN<br>5 => NUMBER<br>; =>SC<br>VALID VARIABLE DECLARATION | Success |

**Conclusion:**

Thus we have successfully implemented the syntax analysis phase of the compiler to validate type and syntax of variable declaration in JAVA using YACC specifications.