Name: Shardul Vikram Dharmadhikari
Roll no: 8016

ASSIGNMENT_6

Problem Statement:

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

1.1 Prerequisite:

Basic concepts of Yacc

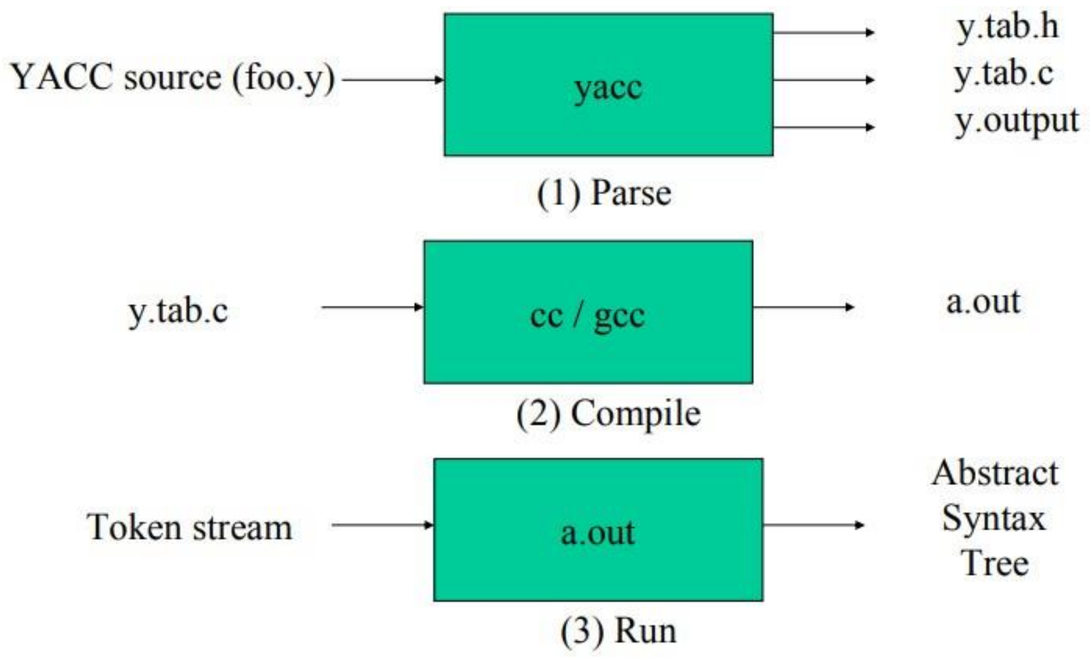
1.2 Learning Objectives:

Understand the implementation Yacc Specification

1.3 YACC(YET ANOYHER COMPILER COMPILER):-

Yacc (for "yet another compiler compiler.") is the standard parser generator for the Unix operating system. An open source program, yacc generates code for the parser in the C programming language. The acronym is usually rendered in lowercase but is occasionally seen as YACC or Yacc. The original version of yacc was written by Stephen Johnson at American Telephone and Telegraph (AT&T). Versions of yacc have since been written for use with Ada, Java and several other less well-known programming languages

1.4 Structure of a Yacc:-



1.4.1 Yacc File Format:-

```
%{
C
declarations
%}
 yacc
declarations
%%
 Grammar rules
 %%
        Additional C code(/*User subroutines */)
```

– Comments in /* ... */ may appear in any of the sections.

YACC Rules:-

- **A grammar rule has the following form:**

A : BODY ; • A is a non-terminal name (LHS). BODY consists of names, literals, and actions. (RHS) literals are enclosed in quotes, eg: '+' '\n' → newline. '\'' → single quote.

Declarations:-

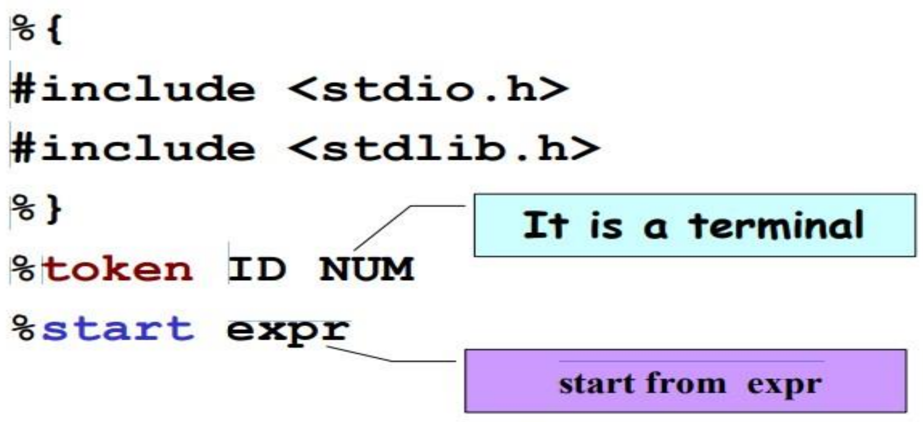
%token : declares ALL terminals which are not literals.
%type : declares return value type for non-terminals.
%union : declares other return types.

The file contains the following sections:

A) Declarations section.

This section contains entries that:

Include standard I/O header file
Define global variables
Define the list rule as the place to start processing
Define the tokens used by the parser
Define the operators and their precedence



```
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token ID NUM
%start expr
```

B) Rules section.

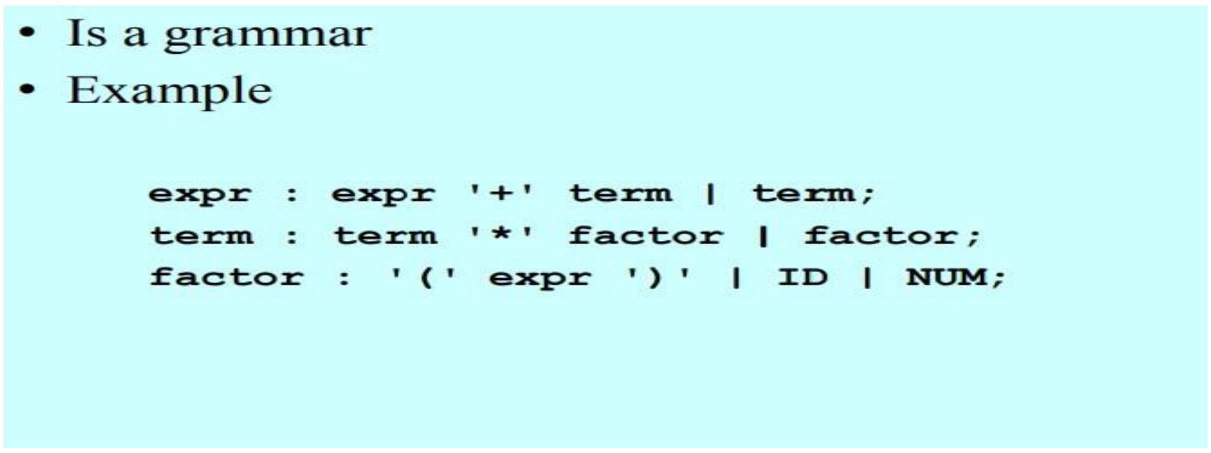
The rules section defines the rules that parse the input stream.

%start - Specifies that the whole input should match **start.**
%union - By default, the values returned by actions and the lexical analyzer are integers. **Yacc** can also support values of other types, including structures. In addition, **yacc** keeps track of the types, and inserts appropriate union member names so that the resulting parser will be strictly type checked. The **yacc** value stack is declared to be a union of the various types of values desired. The user declares the union, and associates union member names to

each token and nonterminal symbol having a value. When the value is referenced through a $$ or $n construction, **yacc** will automatically insert the appropriate union name, so that no unwanted conversions will take place.

%type - Makes use of the members of the **%union** declaration and gives an individual type for the values associated with each part of the grammar.

%token - Lists the tokens which come from lex tool with their type.



- Is a grammar
- Example

```
expr   :  expr  '+'  term  |  term;
term   :  term  '*'  factor  |  factor;
factor :  '('  expr  ')'  |  ID  |  NUM;
```

C) Programs section:-

The programs section contains the following subroutines. Because these subroutines are included in this file, you do not need to use the yacc library when processing this file.

Subroutine	**Description**
main	The required main program that calls the **yyparse** subroutine to start the program.
yyerror(s)	This error-handling subroutine only prints a syntax error message.
yywrap	The wrap-up subroutine that returns a value of 1 when the end of input occurs.

Code:

```
class abc
{
```

```java
public static void main(String args[])
{
int x=60;
//commmm
char c='s';
String s="xyz_@12";
float f=25.25;
a+=f;
c&&s;
System.out.println(a);
}
}
```