ASSIGNMENT 4

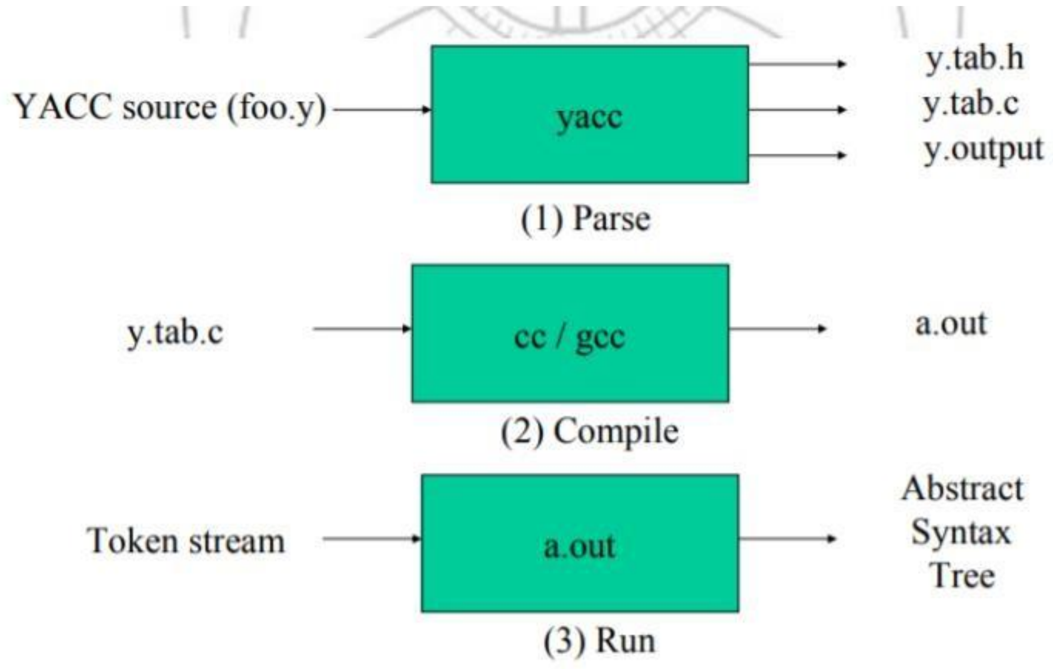
Name – Masood Ismail Tamboli
Roll no – 223081
GR.No - 22020190
Batch – C3

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

Theory:

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

Structure of a Yacc:-



Yacc File Format:-

%{

C declarations

%}

yacc declarations

%%

Grammar rules

%%

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

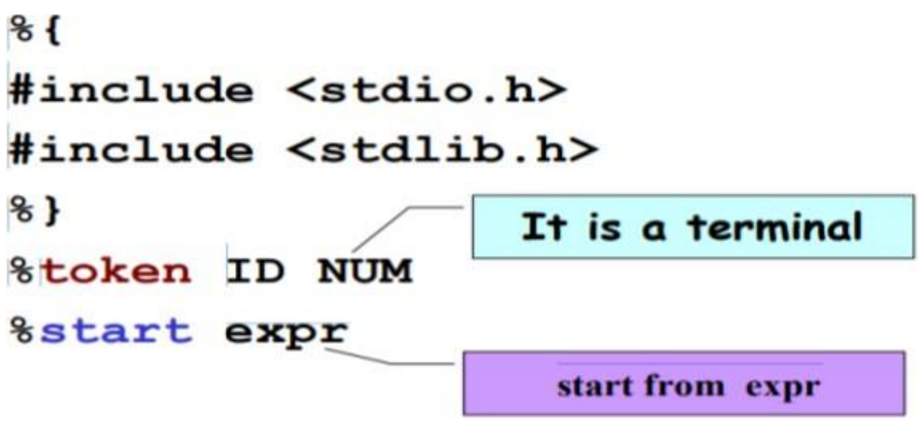
types. The file contains the following

sections:

A) <u>Declarations section</u>

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

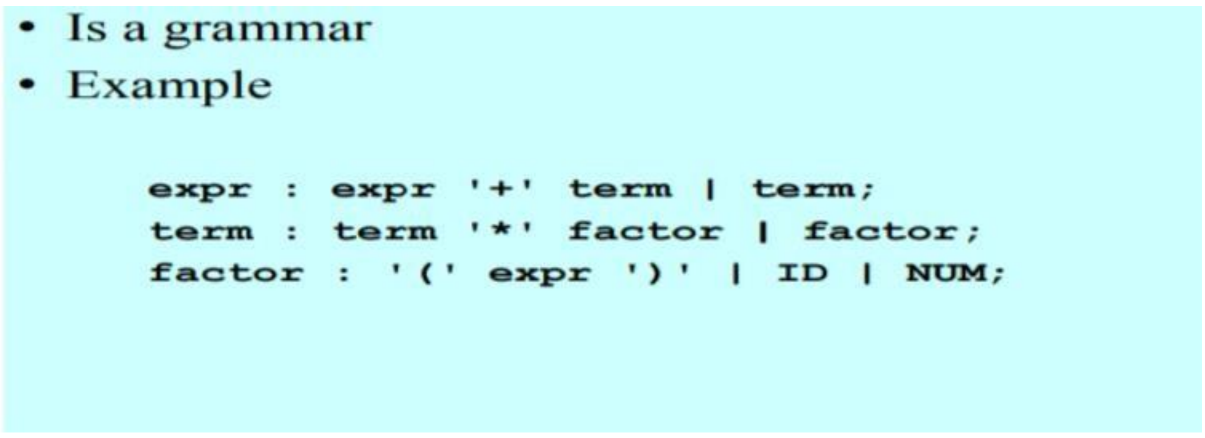


```
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token ID NUM
%start expr
```

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**
1. Main 2. yyerror(s) 3. yywrap	1. The required main program that calls the subroutine to start the program. 2. This error-handling subroutine only prints a syntax error message. 3. The wrap-up subroutine that returns a value of 1 when the end of input occurs.

Code: -

Declare.l (LEX File):

```
%{
#include
<stdio.h>
#include
"y.tab.h"
%}
DIGIT [0-9]
LETTER [A-
Za-z] ASSIGN
=
%%
[\t ] ;
int {printf("%s\t==> DataType\n",yytext);return (INT);}
float {printf("%s\t==> DataType\n",yytext);return
(FLOAT);} char {printf("%s\t==>
DataType\n",yytext);return (CHAR);} boolean
{printf("%s\t==> DataType\n",yytext);return (BL);}
```

```
true|false { printf("%s\t==> BOOLEAN VAL\n",yytext);return
BLVAL;}       ['][^\t\n][']       {       printf("%s\t==>       CHAR
VALUE\n",yytext);return     CHVAL;}     [a-zA-z]+[a-zA-z0-9_]*
{printf("%s\t==> ID\n",yytext);return ID;}
{DIGIT}+ { printf("%s\t==> INT NUMBER\n",yytext);return
NUM;} "," {printf("%s\t==> COMMA\n",yytext);return
COMMA;}
";" {printf("%s\t==> SemiColon\n",yytext);;return SC;}
{ASSIGN} {printf("%s\t==> ASSIGN\n",yytext);return AS;}
\n return NL;
. ;
%%
int yywrap()
{
return 1;
}
```

Declare.y (YACC File):

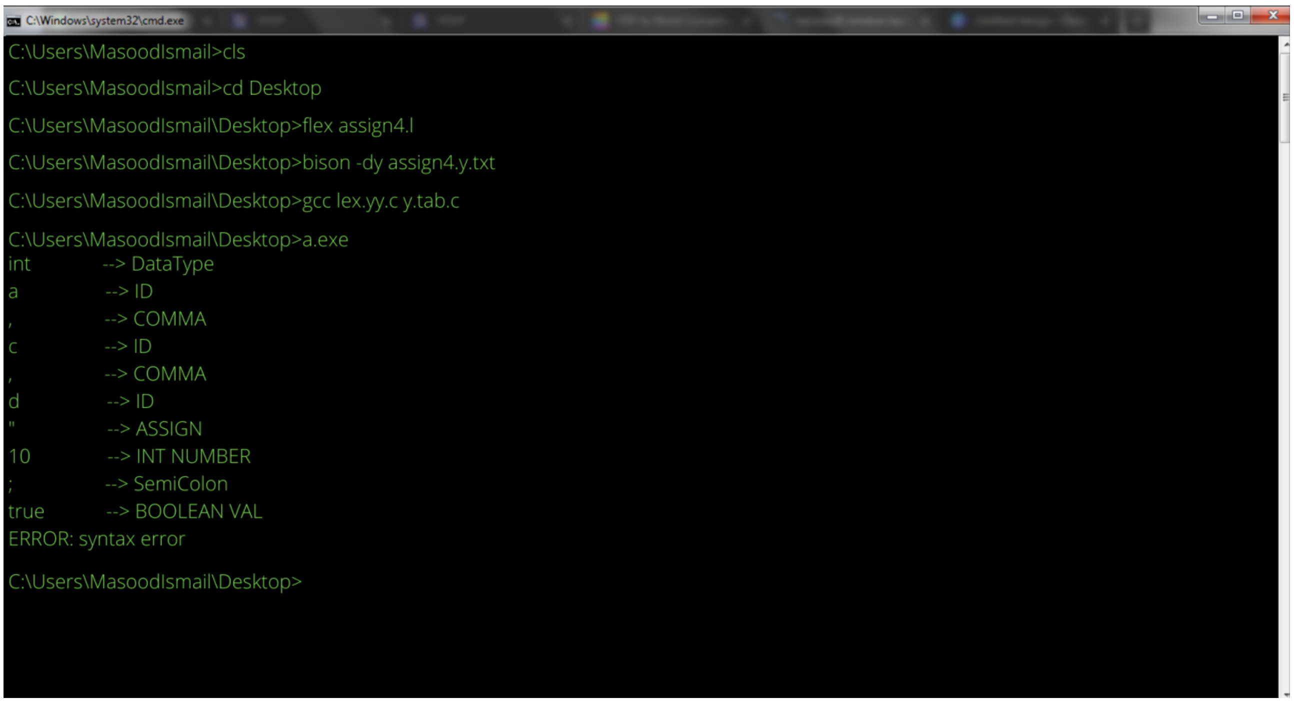
```
%{
#include<stdio.h>
void
yyerror(char*); int
yylex();
FILE* yyin;
%}
%token ID SC INT CHAR FLOAT BL BLVAL CHVAL AS NUM COMMA NL

%%
s: type1|type3|type4
;
type1:INT varlist SC NL { printf("valid INT Variable declaration"); return 0;}
/// for "int a" Test case(without SC ;) NL is added at end otherwise it waits for input
;
type3:CHAR varlist3 SC NL{ printf("valid CHAR Variable declaration");return 0;}
;
type4:BL varlist4 SC NL{ printf("valid BOOLEAN Variable declaration");return 0;}
;

varlist: ID | ID COMMA varlist | ID AS NUM |ID AS NUM COMMA varlist | //THIS IS
FOR EPSILON CASE (EMPTY)
;
varlist3: ID | ID COMMA varlist3 | ID AS CHVAL | ID AS CHVAL COMMA varlist3 |
;
varlist4: ID | ID COMMA varlist4 | ID AS BLVAL | AS BLVAL COMMA varlist4 |
;
%%
```

```
void yyerror(char *s )
{
fprintf(stderr, "ERROR: %s\n",s);
}
int main()
{
        yyin=fopen("input.txt","r")
        ; yyparse();
fclose(yyin);
        return 0;
}
```

Output: -



```
C:\Users\MasoodIsmail>cls

C:\Users\MasoodIsmail>cd Desktop

C:\Users\MasoodIsmail\Desktop>flex assign4.l

C:\Users\MasoodIsmail\Desktop>bison -dy assign4.y.txt

C:\Users\MasoodIsmail\Desktop>gcc lex.yy.c y.tab.c

C:\Users\MasoodIsmail\Desktop>a.exe
int           --> DataType
a             --> ID
,             --> COMMA
c             --> ID
,             --> COMMA
d             --> ID
"             --> ASSIGN
10            --> INT NUMBER
;             --> SemiColon
true          --> BOOLEAN VAL
ERROR: syntax error

C:\Users\MasoodIsmail\Desktop>
```