SSC LAB ASSIGNMENT NO.7
COMPOUND STATEMENT VALIDATION

NAME : KETAKI PATIL
ROLL NO : PA-17
BATCH : A1

Title: Validation of compound statement.

Problem Statement: Write a program using LEX and YACC to validate compound statements in High Level Language

Objective:

1. To study YACC tools for syntax analysis.
2. To master YACC utility.

Theory:

1. **Write about the syntax analysis phase of the compiler.**

Syntax Analysis or Parsing is the second phase, i.e. after lexical analysis. It checks the syntactic structure of the given input, i.e. whether the given input is in the correct syntax (of the language in which the input has been written) or not. It does so by building a data structure, called a Parse tree or Syntax tree. The parse tree is constructed by using the predefined Grammar of the language and the input string. If the given input string can be produced with the help of the syntax tree (in the derivation process), the input string is found to be in the correct syntax. If not, an error is reported by a syntax analyzer.

2. **Description of Each Section of *.y file with example-**

```
YACC input file is divided into three parts.

/* definitions */

 ....


%%
```

```
/* rules */

....

%%

/* auxiliary routines */

....
```

Input File: Definition Part:
Yacc automatically assigns numbers for tokens, but it can be overridden by

```
%token NUMBER 621
```

- Yacc also recognizes single characters as tokens. Therefore, assigned token numbers should not overlap ASCII codes.

- The definition part can include C code external to the definition of the parser and variable declarations, within **%{** and **%}** in the first column.

- It can also include the specification of the starting symbol in the grammar:

```
%start nonterminal
```

Input File: Rule Part:

- The rules part contains grammar definition in a modified BNF form.

- Actions is C code in { } and can be embedded inside (Translation schemes).

Input File: Auxiliary Routines Part:

- The auxiliary routines part is only C code.

- It includes function definitions for every function needed in rules part.

- It can also contain the main() function definition if the parser is going to be run as a program.

- The main() function must call the function yyparse().

Input File:

If yylex() is not defined in the auxiliary routines sections, then it should be included:

#include "lex.yy.c"

- YACC input file generally finishes with:

.y

Output Files:

- The output of YACC is a file named **y.tab.c**
- If it contains the **main()** definition, it must be compiled to be executable.
- Otherwise, the code can be an external function definition for the function **int yyparse()**
- If called with the **–d** option in the command line, Yacc produces as output a header file **y.tab.h** with all its specific definition (particularly important are token definitions to be included, for example, in a Lex input file).
- If called with the **–v** option, Yacc produces as output a file **y.output** containing a textual description of the LALR(1) parsing table used by the parser. This is useful for tracking down how the parser solves conflicts.
- **Example:**

Yacc File (.y)

```
%{
   #include <ctype.h>
   #include <stdio.h>
   #define YYSTYPE double /* double type for yacc stack */
%}
%%
 Lines :  Lines S '\n' { printf("OK \n"); }
       |  S '\n'
       |  error '\n' {yyerror("Error: reenter last line:");
                      yyerrok; };
```

```
 S    :  '(' S ')'
        |  '[' S ']'
        |   /* empty */     ;
%%
#include "lex.yy.c"
void yyerror(char * s)
/* yacc error handler */
{
 fprintf (stderr, "%s\n", s);
}
int main(void)
 {return yyparse();}
```

Lex File (.l)

```
%{
%}
%%
[ \t]      { /* skip blanks and tabs */ }
\n|.       { return yytext[0]; }
%%
```

3. **Description of standard inbuilt variables and functions like yylval,yyparse(),yyerror().**

yylval():

The function returns an integer, the token number, representing the kind of token read. If a value is associated with that token, it should be assigned to the external variable yylval . The parser and the lexical analyzer must agree on these token numbers in order for communication between them to take place.

yyparse():

yyparse() returns a value of 0 if the input it parses is valid according to the given grammar rules. It returns a 1 if the input is incorrect and error recovery is impossible. **yyparse**() does not do its own lexical analysis. In other words, it does not pull the input apart into tokens ready for parsing.

yyerror():

The first thing the parser does when it performs the **Error** action is to call a function named **yyerror()**. This happens before the parser begins going down the state stack in search of a state that can handle the **error** symbol. **yyerror()** is a **lex** and **yacc** library function that simply displays a text string argument to **stderr** using **fprintf**, and returns the integer value received from **fprintf**.

4. **Compilation and Execution Process**
    1. Write lex program in a file file.l and yacc in a file file.y
    2. Open Terminal and Navigate to the Directory where you have saved the files.
    3. type flex filename.l
    4. type bison -d filename.y
    5. type gcc lex.yy.c
    6. type a.exe

Input: Source specification (*.y) file for loop statement like If statements, while and do-while statements.

```
if (a<b) a=9;

while(a<b) a=9;
```

Output: statement is grammatically correct or not.

```
if(a<b) a=9;

VALID SINGLE STATEMENT IF

while(a<b) a=9;

VALID SINGLE STATEMENT WHILE LOOP
```

Conclusion: Parser for validation of compound statement is successfully done.

Platform: Linux (Lex and Yacc)

FAQs:

1. What is the role of the y.tab.h file?

Before writing the LEX program, there must be some way by which the YACC program can tell the LEX program that DIGIT is a valid token that has been declared in the YACC program. This communication is facilitated by the **file** "**y**. **tab**. **h**" which contains the declarations of all the tokens in the YACC program.

2. Explain YACC tool.

- 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 input of YACC is the rule or grammar and the output is a C program.

PROGRAM :

```
/*
 SSC LAB ASSIGNMENT NO.7
COMPOUND STATEMENT
NAME : KETAKI PATIL
ROLL NO : PA-17
BATCH : A1
 */
```

LEX FILE :

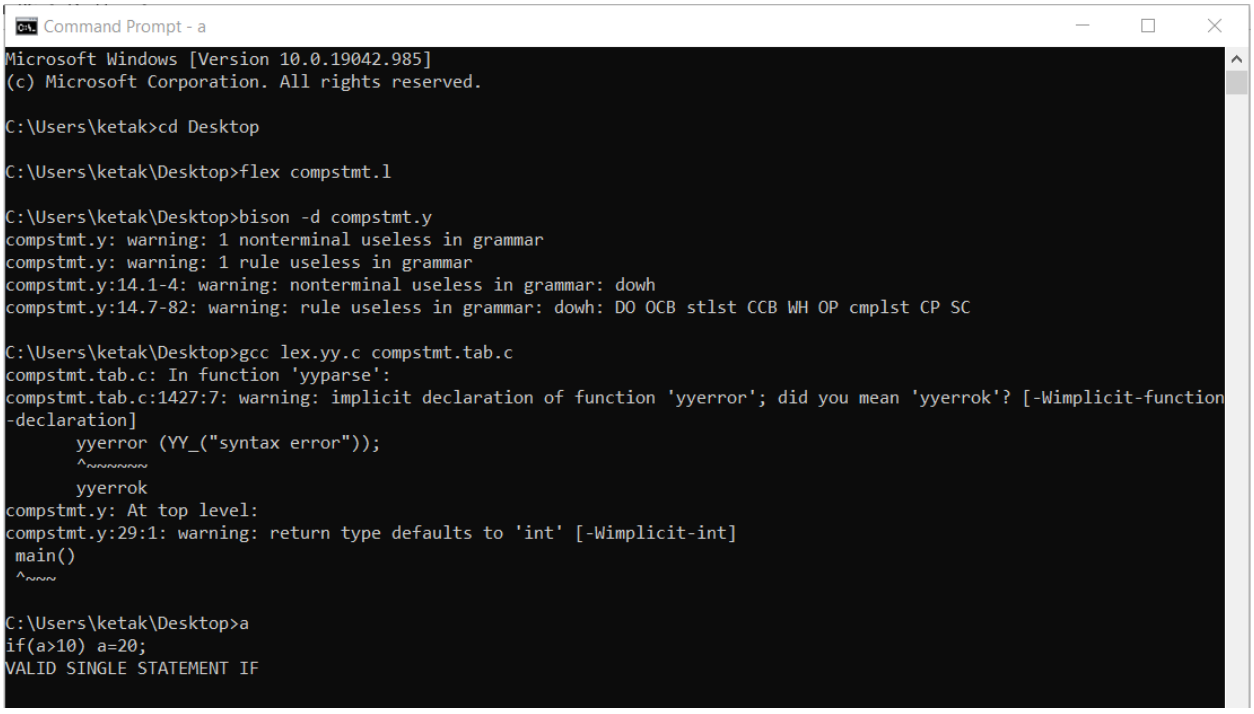
```
%{
    #include "compstmt.tab.h"
    extern int yyerror(char *str);
    extern int yyparse();
%}

%%
"while"     return WH;
"if"        return IF;
"do"        return DO;
"for"       return DO;
"("         return OP;
")"         return CP;
"{"         return OCB;
"}"         return CCB;
"<" |
">" |
"<=" |
">=" |
"==" |
"!=" return CMP;
"+" |
"-" |
"*" |
"/" return OPR;
"=" return ASG;
([a-zA-Z])("_"|[a-zA-Z0-9])* return ID;
[0-9]+ return NUM;
";" return SC;
"," return COMMA;
" " {}

%%
int yywrap()
{
    return 1;
}
```

YACC FILE :

```
%{
#include<stdio.h>
extern int yylex();
extern int yywrap();
extern int yyparse();
%}

%token WH IF DO FOR OP CP OCB CCB CMP SC ASG ID NUM COMMA OPR

%%
start:  swh | mwh | sif | mif;
swh:  WH OP cmplst CP stmt                      {printf("VALID SINGLE
STATEMENT WHILE LOOP\n");};
```

```
mwh:  WH OP cmplst CP OCB stlst CCB            {printf("VALID MULTI
STATEMENT WHILE LOOP\n");};
dowh: DO OCB stlst CCB WH OP cmplst CP SC      {printf("VALID DO-WHILE
LOOP\n");};
sif:  IF OP cmplst CP stmt                     {printf("VALID SINGLE
STATEMENT IF\n");};
mif:  IF OP cmplst CP OCB stlst CCB            {printf("VALID MULTI
STATEMENT IF\n");};
cmplst: cmpn COMMA cmplst | cmpn;
cmpn: ID CMP ID | ID CMP NUM;
stlst:  stmt stlst | stmt;
stmt: ID ASG ID OPR ID SC | ID ASG ID OPR NUM SC | ID ASG NUM OPR ID SC |
ID ASG NUM OPR NUM SC | ID ASG ID SC | ID ASG NUM SC
      | start               {printf("NESTED INSIDE A");}

%%
int yyerror(char *str)
{
      printf("%s", str);
}

main()
{
      yyparse();
}
```

OUTPUT SS :



```
C:\Users\ketak\Desktop>a
while(i<10) i=5;
VALID SINGLE STATEMENT WHILE LOOP


C:\Users\ketak\Desktop>a
if(a>10) a=20;
VALID SINGLE STATEMENT IF


C:\Users\ketak\Desktop>a
while(a>b) {a=5; b=6;}
VALID MULTI STATEMENT WHILE LOOP


C:\Users\ketak\Desktop>a
if(a>b) {a=45; b=20;}
VALID MULTI STATEMENT IF
```