Name: Sanyukta Tambankar.
Roll: PA 58 Batch: A4

Lab Assignment 7.

* Title: Validation of compound statement

* Problem Statement: Write program using LEX & YACC to validate compound statements in high level language.

* Objective:
  1) To study YACC tools for syntax analysis.
  2) Master YACC utility.

* Theory:

1) Write about syntax analysis of compiler.

→ It's second phase after lexical analysis. It checks the syntactical structure of given input. It does so by building data structure called parse tree or syntax tree. Parse tree is constructed by using the predefine Grammar of language & i/p string.

2) Description of standard inbuilt variables & functions like yylval, yyparse(), yyerror().

→ yylval() ⇒ Function returns an integer, the token number, representing kind of token read. If value is associated with token it should to external variable yylval.

yyparse() ⇒ Returns a value of 0 if input it parse is valid according to given grammar rules.

yyerror() ⇒ First thing the parser does when it performs error action is to call yyerror(). This happens before parser begins going down the state stack in search of state that can handle error symbol.

3) Compilation & Execution Process:

i Write Lex algorithm in file.l & file.y.

ii Open terminal & navigate directory where you saved files

iii type flex filename.l

iv type bison -d filname.y

v type gcc lex.yy.c

vi type a.exe.

* Input: - .y file for loop statements.

* Output: Statement is correct or not.

* Conclusion: Parser for validation of compound statement.

* FAQs.

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

Ans: Before writing LEX program there must be some way by which YACC program can tell LEX program the DIGIT is valid token has been declared in YACC program.

2) Explain YACC Tool.

Ans: YACC stands for Yet Another compiler compiler.

- YACC provides tool to produce a parser for given grammar.
- YACC is designed to compile LALR
- The input of YACC rule or grammar & output is C program.

Name: Sanyukta Tamhankar

RollNo: PA58

Batch A4.

Code and Output:

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

%%

"while" return WH;
"if" return IF;
"do" return DO;
"for" return FOR;
"(" 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;

}


%{

#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 | dowh | 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();
}
```