UCS 1602 - Compiler Design

Exercise 6 - Implementation of Syntax checker using Lex and Yacc Tools

Name:	Mahesh Bharadwaj K
Reg No:	185001089
Semester:	VI
Date:	March 29, 2021

Aim:

To implement Syntax checker using Lex and Yacc Tools

Program

Lexer file

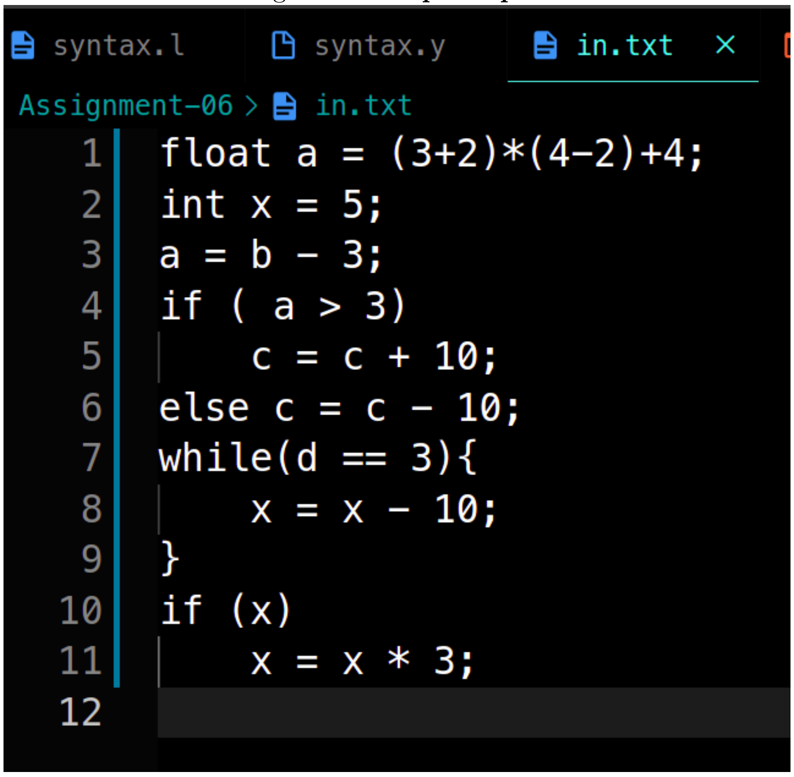
```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include "y.tab.h"
    int debug=0;
%}
%%
[ \t\n]+ ;
[0-9]+ { if(debug)printf("%s is an integer\n", yytext);return INT;}
("int"|"float"|"char"|"double") {if(debug)printf("%s is a data type\n", yytext);return
↪  DTYPE; }
"if" {return IF;}
"while" {return WHILE;}
"else" {return ELSE;}
[_a-zA-Z][a-zA-Z0-9_]* { if(debug)printf("%s is an identifier\n", yytext);return ID; }
";" {if(debug)printf("End of statement\n");return EOS;}
(">"|"<"|"<="|">="|"!="|"==") { return COMPARSION_OP; }
("+="|"-="|"*="|"/="|"=") { if(debug)printf("%s is an assign op\n", yytext);return
↪  ASSIGN_OP; }
">>" { return RSHIFT; }
"<<" {return LSHIFT; }
"!" { return NOT; }
"{" { return *yytext; }
"}" { return *yytext; }
"||" {return OR; }
"&&" {return AND; }
[+\-^*/,().] {return *yytext;}
. {
  fprintf(stderr,"Unknown token found: <%s>\n", yytext);
}
%%
```

Yacc Program

```
%{
    #include <stdlib.h>
    #include <stdio.h>
    int yylex(void);
    extern FILE* yyin;
    #include "y.tab.h"
    int error = 0;
    /*extern int debug;*/
%}
%token INT ASSIGN_OP COMPARSION_OP ID DTYPE LSHIFT RSHIFT NOT AND OR EOS IF ELSE WHILE

%%
program : statement EOS programPrime
        | loop_block programPrime
        | cond_statement programPrime
        ;

programPrime:   program
            |
            ;
statement: declaration
        | expr  {printf("Expression found!\n");}
        | ID ASSIGN_OP expr {printf("Expression found!\n");}
        | cond_statement
        ;
cond_statement  : IF '(' expr ')' statement EOS optional
                ;
optional    : ELSE statement EOS{printf("IF with else!\n");}
            |                   {printf("IF without else!\n");}
            ;

loop_block  : WHILE '(' expr ')' '{' statement EOS loop_optional
            ;
loop_optional   : '}' {printf("While loop found!\n");}
                | statement EOS
                | loop_optional
                ;

declaration : DTYPE ID ASSIGN_OP INT { printf("Declaration with assignment
↪   found!\n");}
            | DTYPE ID {printf("Declaration found!\n");}
            | DTYPE ID ASSIGN_OP expr {printf("Declaration with expr found!\n");}
            ;

expr    : expr '+' expr
        | expr '-' expr
        | expr '*' expr
        | expr '/' expr
        | expr '^' expr
        | expr AND expr
        | expr OR expr
        | NOT expr
        | '(' expr ')'
        | expr LSHIFT expr
        | expr RSHIFT expr
        | expr COMPARSION_OP expr
        | INT
        | INT '.' INT
        | ID
        ;
%%
int yyerror(){
    fprintf(stderr, "Syntax is NOT valid!\n");
    error = 1;
    return 0;
}

int yywrap(){
    return 1;
}

int main(int argc, char **argv){
    /*yydebug = 1;*/
    if(argc != 2){
        fprintf(stderr, "Enter file name as argument!\n");
        return 1;
    }
    yyin = fopen(argv[1], "rt");
    if (!yyin){
        fprintf(stderr, "File not found!\n");
        return 2;
    }
    yyparse();
    if(!error){
        printf("Valid syntax!\n");
    }
    return 0;
}
```

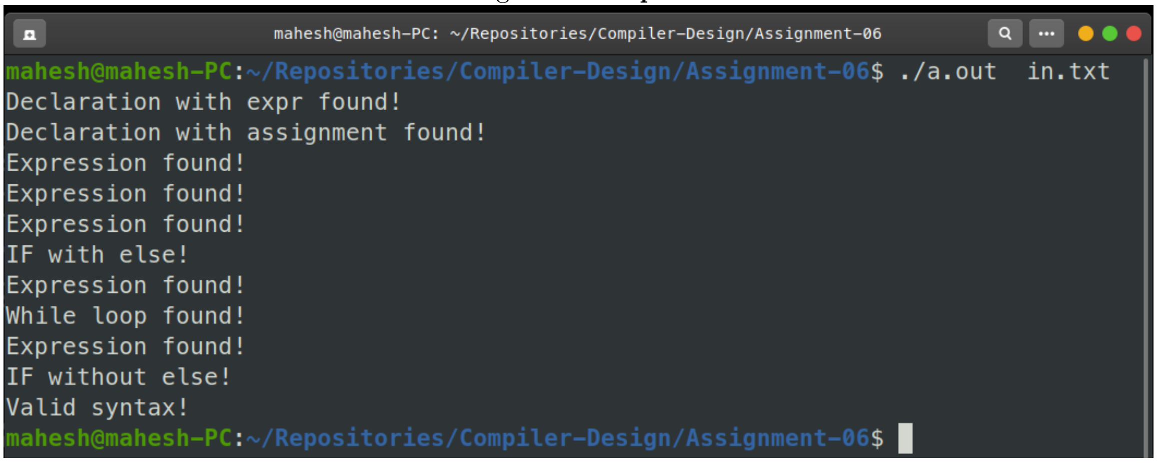
Output

Figure 1: **Sample Input**



```
float a = (3+2)*(4-2)+4;
int x = 5;
a = b - 3;
if ( a > 3)
    c = c + 10;
else c = c - 10;
while(d == 3){
    x = x - 10;
}
if (x)
    x = x * 3;
```

Figure 2: **Output**



mahesh@mahesh-PC: ~/Repositories/Compiler-Design/Assignment-06

```
mahesh@mahesh-PC:~/Repositories/Compiler-Design/Assignment-06$ ./a.out  in.txt
Declaration with expr found!
Declaration with assignment found!
Expression found!
Expression found!
Expression found!
IF with else!
Expression found!
While loop found!
Expression found!
IF without else!
Valid syntax!
mahesh@mahesh-PC:~/Repositories/Compiler-Design/Assignment-06$
```

Learning Outcomes

1. We learn to write grammar for expressions.
2. We learn to Write rules to parse tokens in grammar.
3. We learn to create syntax analyser using YACC and LEX tool.