Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VI

28 March 2021

UCS 1602 - Compiler Design

Exercise 6: Implementation of Syntax Checker Using Yacc Tool

Aim:

Develop a **Syntax Checker** to recognize the tokens necessary for the following statements by writing suitable grammars.

- Assignment Statement
- Conditional Statement
- Looping Statement

Code - Yacc Parser File:

```
%{
    #include <stdio.h>
    #define YYSTYPE double
    int flag = 0;
%}

%token  NUM ASSIGN ID
%token  RELOP LOGIC ARITH INCDEC
%token  IF ELIF ELSE
%token  FOR WHILE

%%
Block   :   Stmt Block
        |   Stmt
        ;

Stmt    :   Loop '{' Block '}'
        |   ConStmt '{' Block '}'
        |   Expr ';'

Loop    :   FOR '(' Expr ';' Condns ';' Expr ')'
        |   FOR '(' ';' Condns ';' ')'
        |   WHILE '(' Condns ')'
        ;

ConStmt :   IF '(' Condns ')'
        |   ELIF '(' Condns ')'
        |   ELSE
        ;

Condns  :   Condn LOGIC Condns
        |   Condn
        ;

Condn   :   ID RELOP ID
        |   ID RELOP NUM
        |   ID
        ;

Expr    :   Init
        |   ID ASSIGN ID ARITH ID
        |   ID ASSIGN ID ARITH NUM
        |   ID ASSIGN NUM ARITH NUM
        |   ID INCDEC
        |   INCDEC ID
        ;

```

```
Init    :   ID ASSIGN Init
        |   ID ASSIGN ID
        |   ID ASSIGN NUM
        ;
%%

int yyerror(char *s){
    flag = 1;
    //fprintf(stderr, "%s\n", s);
    return 1;
}

int main(void){
    printf("\n\n\t\tSYNTAX CHECKER USING YACC\n");
    printf("\nNote: Enter the code snippet in Code.txt.\n");
    printf("\nCode Obtained:\n\n");
    system("cat Code.txt");
    yyparse();

    if(flag){
        printf("\nSyntactically Incorrect.\n");
    }

    else{
        printf("\nSyntactically Correct.\n");
    }

    return 0;
}

/* Usage:
        Run yacc -d Check.y
        Run lex Check.l
        Run gcc lex.yy.c -lm -w
        Run ./a.out < Code.txt
*/
```

Code - Lex Grammar File:

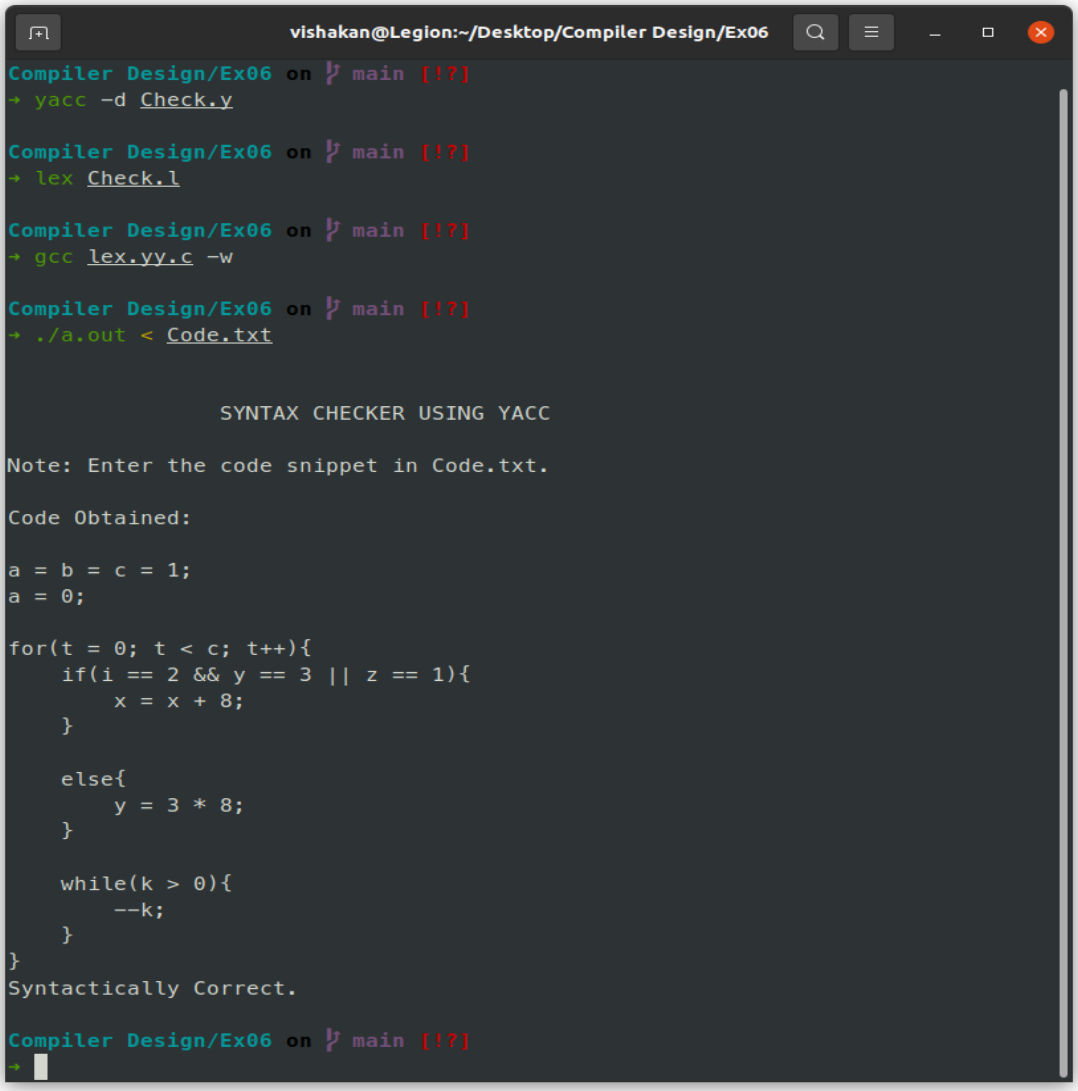
```
%{
    #include <stdio.h>
    #include "y.tab.c"
    extern YYSTYPE yylval;
%}

assign      ("=")
relop       ("=="|"!="|">="|"<="|"<"|">")
arithop     ("+"|"-"|"/"|"%"|"*")
incdec      ("++"|"--")
logical     ("||"|"&&")
identifier  [a-zA-Z_][a-zA-Z0-9_]*


%%

[0-9]+          {return NUM;}
{assign}        {return ASSIGN;}
{relop}         {return RELOP;}
{logical}       {return LOGIC;}
{arithop}       {return ARITH;}
{incdec}        {return INCDEC;}
"if"            {return IF;}
"else if"       {return ELIF;}
"else"          {return ELSE;}
"for"           {return FOR;}
"while"         {return WHILE;}
{identifier}    {return ID;}


[ \t]           {;}
[\n]            {;}
.               {return *yytext;}

%%

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

Sample - Parsed C Code:

```
a = b = c = 1;
a = 0;

for(t = 0; t < c; t++){
    if(i == 2 && y == 3 || z == 1){
        x = x + 8;
        a = b + c;
    }

    else{
        y = 3 * 8;
    }

    while(k > 0){
        --k;
    }
}
```

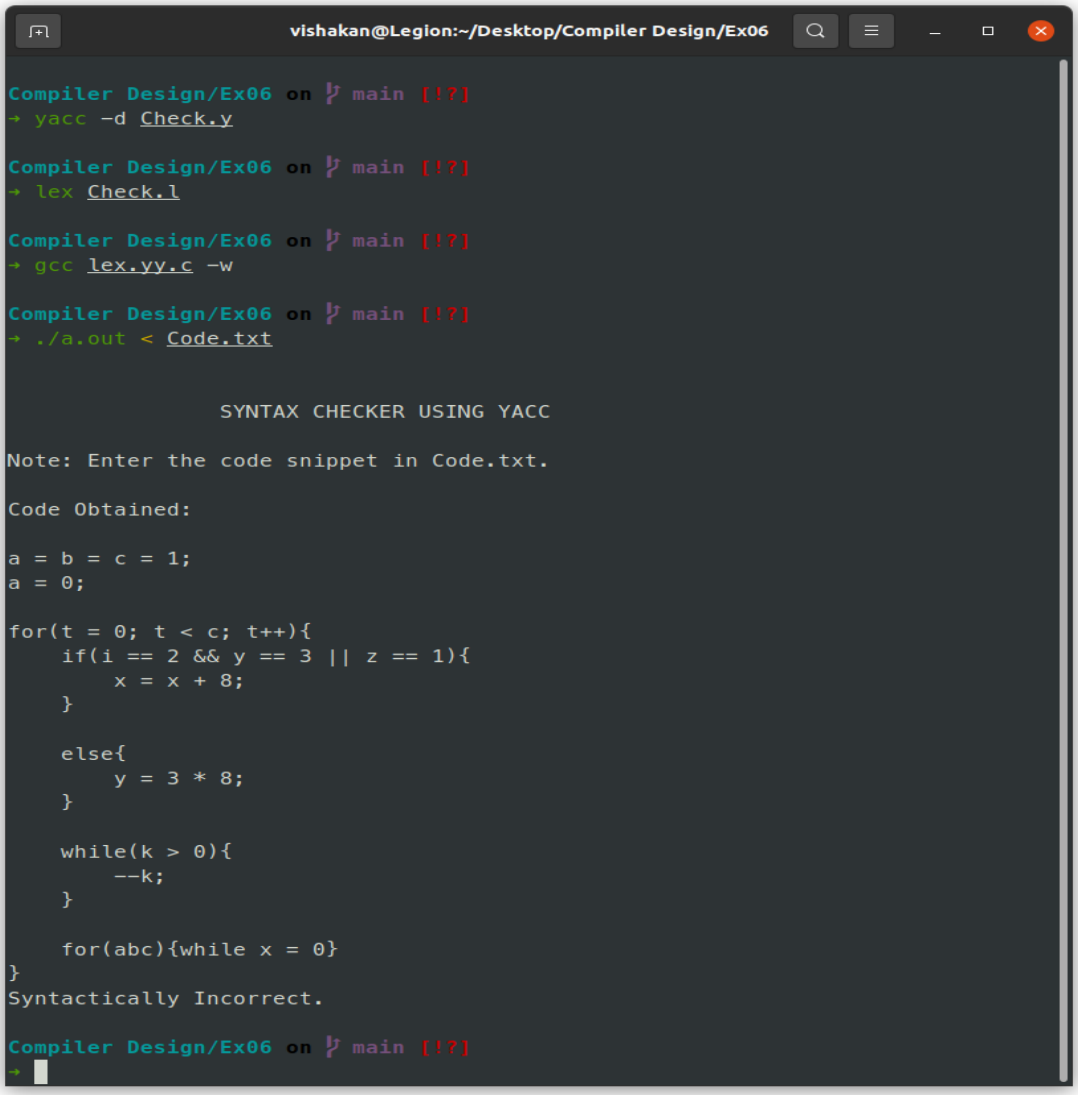
Output 1 - Valid Case:

Figure 1: Console Output - Valid Case.



Output 2 - Invalid Case:

Figure 2: Console Output - Invalid Case.



Learning Outcome:

- I learnt more theory behind **Yacc Parser Generator**.
- I understood how to construct a grammar for a basic syntax checker.
- I learnt that grammar can be built upon layer by layer, each one adding more detail and complexity.
- I learnt that Yacc parser is able to handle Left Recursive grammar as well, since it is a LALR(1) parser.
- I was able to implement the required token recognition with Lex tool.
- I was able to implement a parser with Yacc to mimic the features of a syntax checker.
- I realized key implementation differences between the syntax checker and the desk calculator.
- I learnt how the Yacc parser catches an error using the inbuilt yyerror() function.