SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Engineering

Name: _	Date of Performance: _ _ /_ _/20_ _
Class: B.E. Computer	Date of Completion: _ _ /_ _/20_ _
Division :	Grade:
Batch:	
Roll No:	Sign. of Teacher with Date:
Subject: Compiler Design Lab	

Experiment No. 4

Aim: Implementation of Context Free Grammar

1. Objective: Students will be able to implement any context free grammar.

2. Background:

A context-free grammar (CFG) is a set of recursive rewriting rules (or *productions*) used to generate patterns of strings.

A CFG consists of the following components:

- a set of *terminal symbols*, which are the characters of the alphabet that appear in the strings generated by the grammar.
- a set of *nonterminal symbols*, which are placeholders for patterns of terminal symbols that can be generated by the nonterminal symbols.
- a set of *productions*, which are rules for replacing (or rewriting) nonterminal symbols (on the left side of the production) in a string with other nonterminal or terminal symbols (on the right side of the production). $A \rightarrow \alpha$ where α is any combination of terminals or non-terminals.
- a *start symbol*, which is a special nonterminal symbol that appears in the initial string generated by the grammar.

To generate a string of terminal symbols from a CFG:

- Begin with a string consisting of the start symbol;
- Apply one of the productions with the start symbol on the left hand size, replacing the start symbol with the right hand side of the production;
- Repeat the process of selecting nonterminal symbols in the string, and replacing them with the right hand side of some corresponding production, until all nonterminals have been replaced by terminal symbols.

3. Pre-lab Task:

For any language given generate its equivalent context free grammer.

CFG for $L= \{ 0^n1^n \mid n > 1\}$ can be generated by using following production rules.

$$S \rightarrow 0S1 \mid 01$$

For implementing this CFG, lexical analyzer should pass the token i.e. terminal symbols of a language to parser.

Example : Lex file for above language can be given as :

```
%{
#include "y.tab.h"
%}

%%
"0" {return ZERO;}
"1" {return ONE;}
[\n] {return NL;}
. ;
%%
```

Parser code can be written as:

```
%token ONE ZERO NL

%%

str1: str2 n1 { }
    ;

str2:  ZERO str2 ONE { }
     | ZERO ONE { }
    ;

n1  : NL  { printf("\n The string is accepted");return;}
    ;
%%
main()
{
        yyparse();
}
```

4. In-lab Task:

1. Create lex file (____.l) with definitions of required language. Tokens must be of the terminal symbols of CFG.
2. Create parser file (_____.y) with the actual production rules of CFG.

5. Post-lab Task:

Outcomes:

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Questions:

1. Write the lexer and parser definations and rules to generate a language with equal number of a's and b's.
   L = {ab, ba, abba, baba, bbaa, aabb, ...}