

# CS251: Introduction to Language Processing

## YACC Tutorial

**Vishwesh Jatala**

Department of CSE

Indian Institute of Technology Bhilai

[vishwesh@iitbhilai.ac.in](mailto:vishwesh@iitbhilai.ac.in)

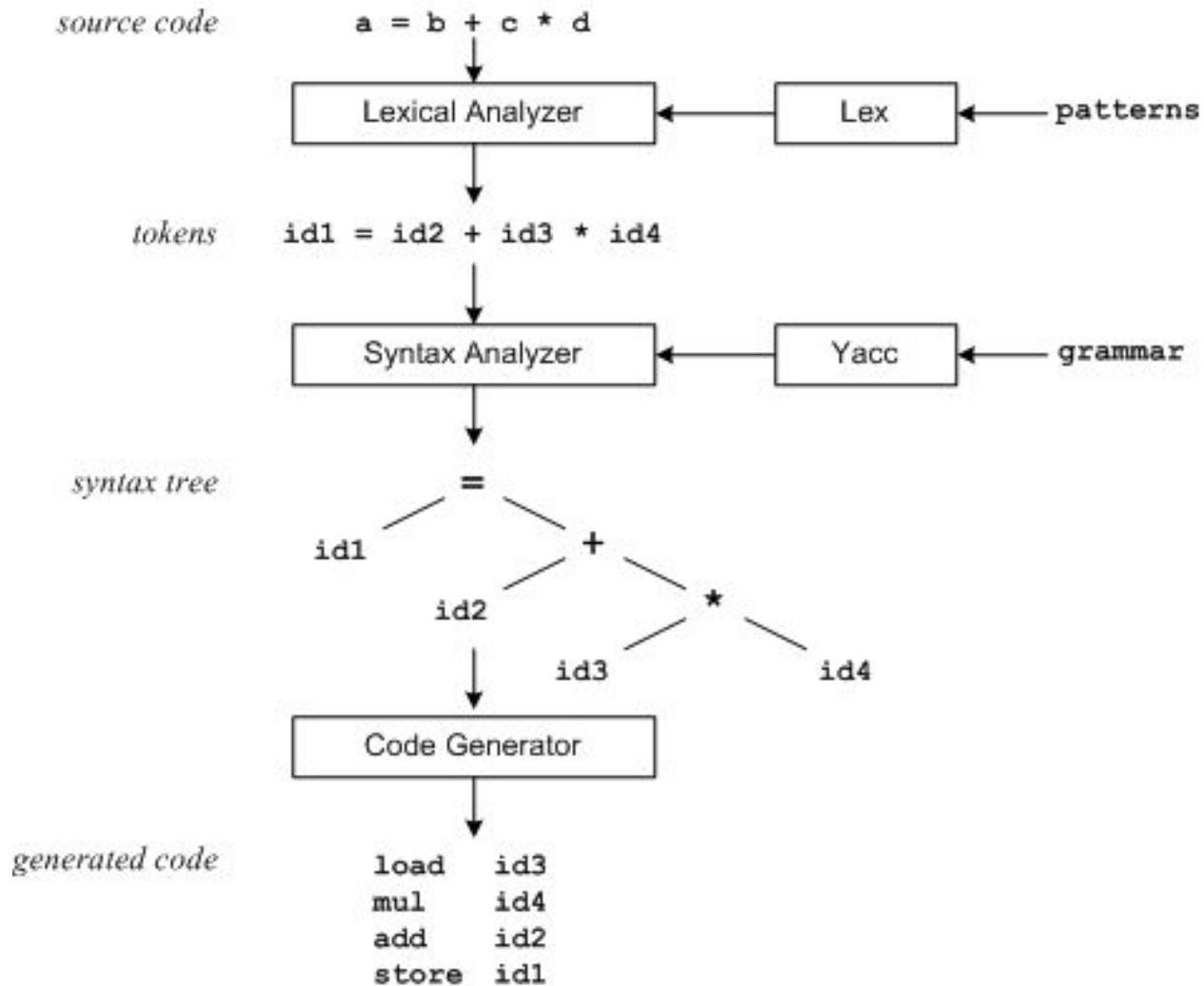


2023-24-M

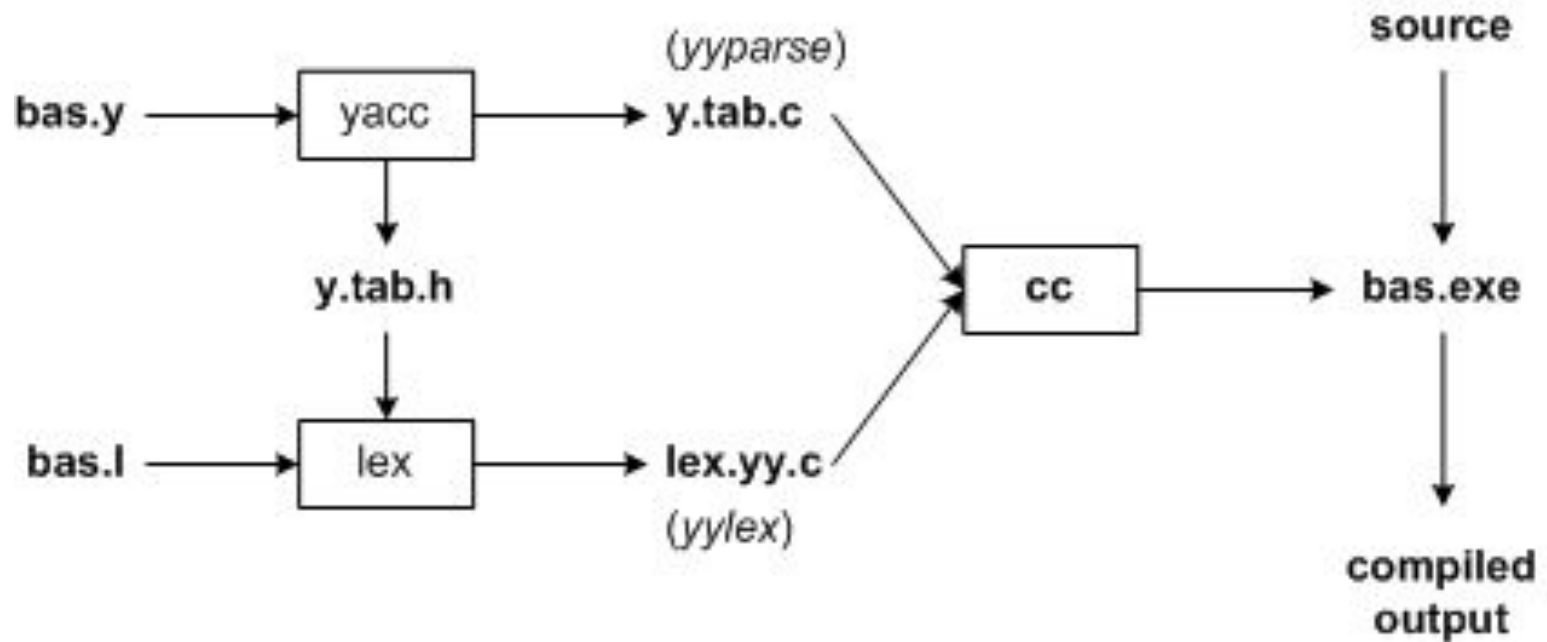
# Parser Generator

- Lex is a tool that generates lexical Analyzer.
- Lexical Analyzer takes input as source code and generate output as tokens
- YACC: Yet Another Compiler Compiler
- It does parsing and semantic processing over the tokens generated through lex.
- Bison: parser generator, the GNU version of Yacc.

# lex / yacc



# Contd...



# Structure of yacc Program

- The yacc program has following structure:

Declarations

%%

Production Rules ACTIONS

%%

Supporting C routines

# Yacc - Declarations (First part)

- The declaration section defines macros and imports header files written in C within `%{ %}`.
- yacc definitions
  - `%start`
  - `%token`
  - `%left`

# Yacc - Productions (Second part)

- It represents a grammar known as set of productions.
- The left hand side of a production is followed by colon (:), and right hand side.
- Multiple right-hand sides may follow separated by a '|'.
- Actions associated with a rule are entered in braces.

# Yacc – Example Productions

```
statements: statement1 {printf("statement1");}  
          | statement1 statements {printf("statements \n");}
```

```
statement1: identifier '+' identifier {printf("plus\n");}  
          | identifier '-' identifier {printf("minus\n");}
```



# Yacc Productions

- $\$1, \$2, \dots, \$n$  can refer to the values associated with symbols.
- $\$\$$  refer to the value of the left.
- Every symbol have a value associated with it (including token and non-terminals)
- Default action :  $\$\$ = \$1$

# Yacc Example productions

statement1: identifier '+' identifier	{ \$\$ = \$1 + \$3; }
identifier '-' identifier	{ \$\$ = \$1 - \$3; }

# Yacc - Third Part

- Contains valid C code that supports the language processing
- Symbol table implementation
- Functions that might be called by actions associated with the productions in the second part

# **SAMPLE PROGRAM**

# Calc.l

```
%{  
    /* Definition section */  
    #include<stdio.h>  
    #include "y.tab.h"  
    extern int yylval;  
%}  
/* Rule Section */  
%%  
[0-9]+ {  
    yylval=atoi(yytext);  
    return NUMBER;  
}  
[\\t]    ;  
[\\n]    return 0;  
.        return yytext[0];  
  
%%  
int yywrap()  
{  
    return 1;  
}
```

# Calc.y

```
%{  
    /* Definition section */  
    #include<stdio.h>  
    int flag=0;  
}%  
%start ArithmeticExpression  
%token NUMBER  
%left '+' '-'  
%left '*' '/' '%'  
%left '(' ')'
```

DEFINITION

```
/* Rule Section */  
%%  
ArithmeticExpression: E{ printf("\nResult=%d\n", $$); return 0; }  
E      :    E+'E' {$$=$1+$3;}  
        |    E-'E' {$$=$1-$3;}  
        |    E'*'E {$$=$1*$3;}  
        |    E/'E' {$$=$1/$3;}  
        |    E%'E' {$$=$1%$3;}  
        |    '('E')' {$$=$2;}  
        |    NUMBER {$$=$1;}  
%%
```

PRODUCTION RULES

# Contd...

```
void main()
{
    printf("\nEnter Any Arithmetic Expression which can
    have operations Addition, Subtraction, Multiplication,
    Division, Modulus and Round brackets:\n");

    yyparse();
    if(flag==0)
        printf("\nEnter arithmetic expression is Valid\n\n");
}

void yyerror()
{
    printf("\nEnter arithmetic expression is Invalid\n\n");
    flag=1;
}
```

FUNCTIONS

Demo!