CSL302: Compiler Design

YACC Tutorial

Vishwesh Jatala

Department of CSE

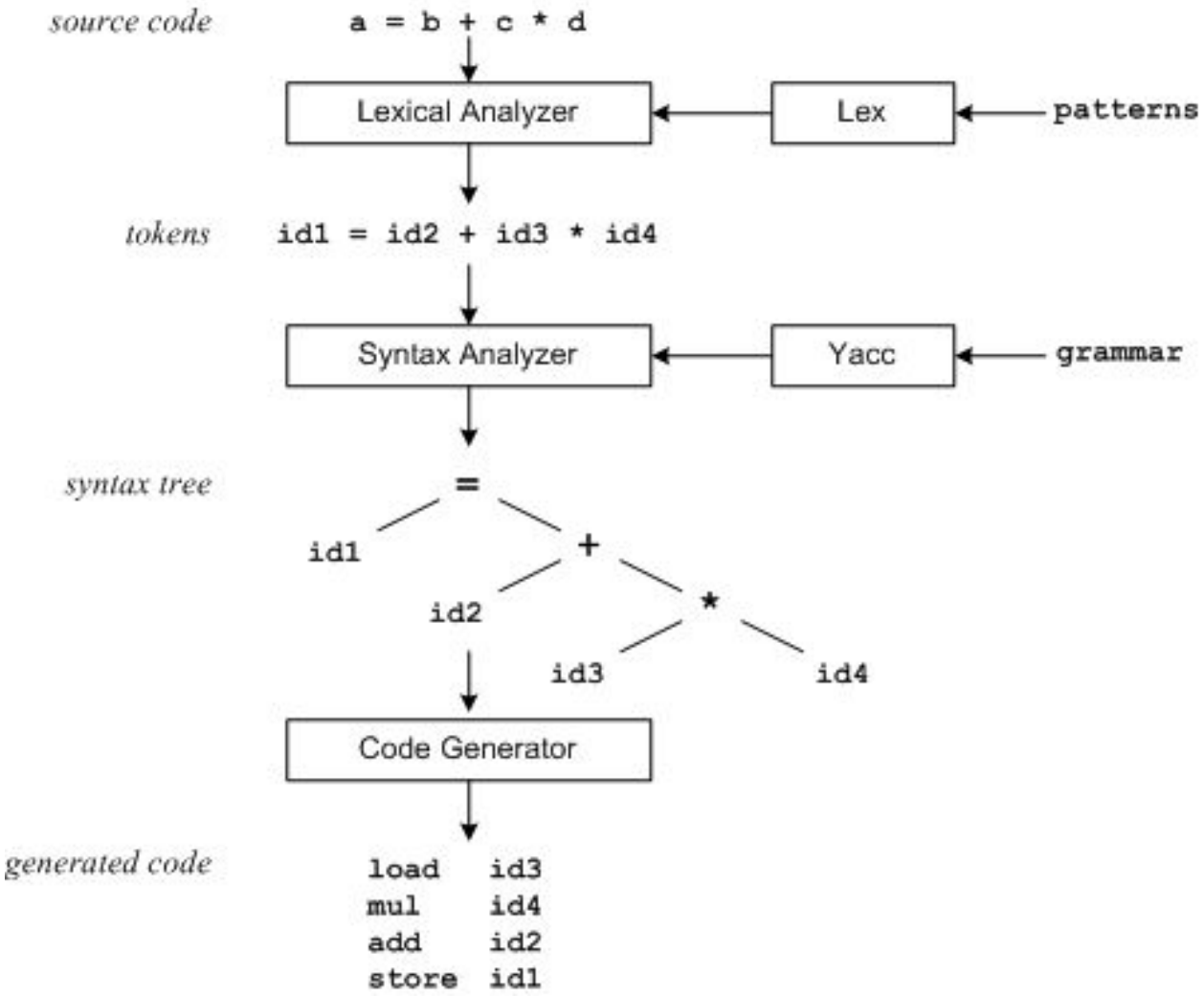
Indian Institute of Technology Bhilai

vishwesh@iitbhilai.ac.in

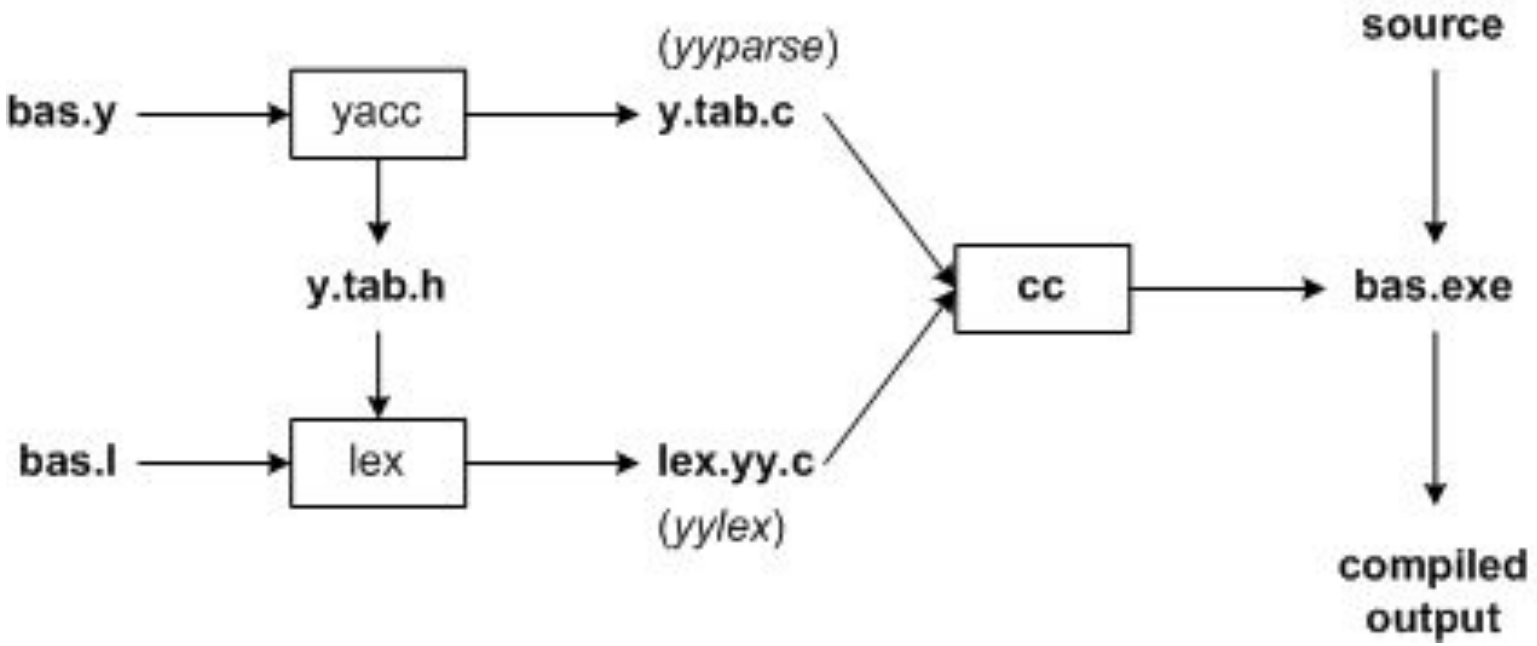
Parser Generator

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

lex / yacc


source code
a = b + c * d
Lexical Analyzer
Lex
patterns
tokens
id1 = id2 + id3 * id4
Syntax Analyzer
Yacc
grammar
syntax tree
=
id1
+
id2
*
id3
id4
Code Generator
generated code
load id3
mul id4
add id2
store id1

Contd...


bas.y
yacc
(yyparse)
y.tab.c
y.tab.h
bas.l
lex
lex.yy.c
(yylex)
cc
source
bas.exe
compiled
output

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, $2,....$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 '(' ')‘

/* 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;}
 %%
```

DEFINITION

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("\nEntered arithmetic expression is Valid\n\n");
}

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

FUNCTIONS

Demo!