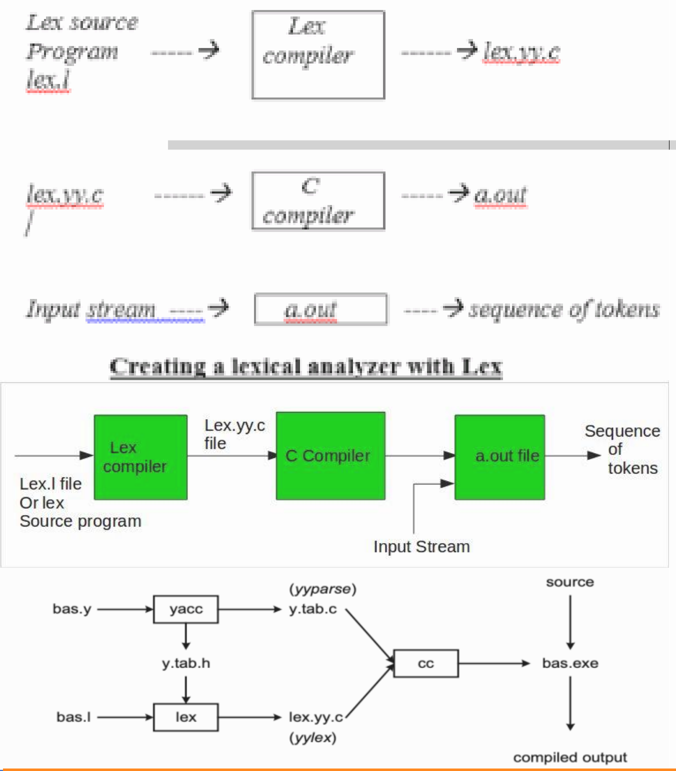
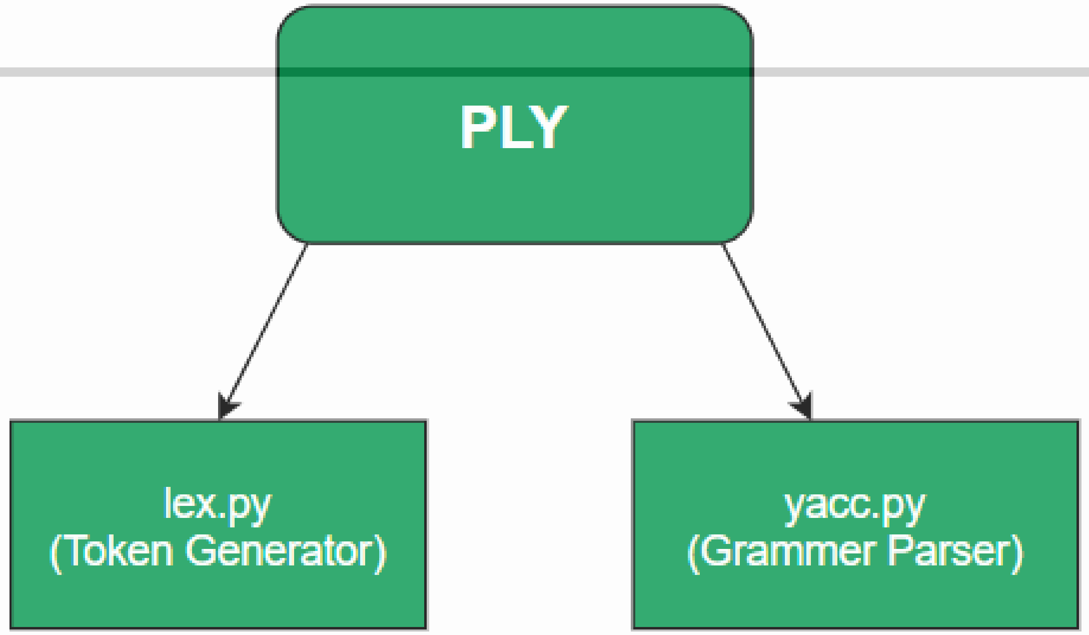
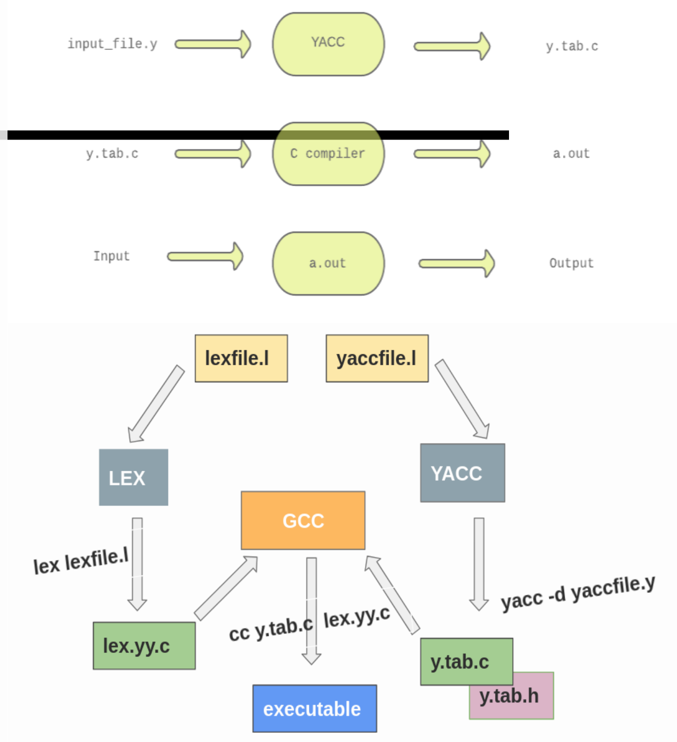






Working with **lex and Yacc**

Seminar - 09

ABOUT LEX AND YACC

Lex is a lexical analyser generator used in **compiler design** to convert a sequence of **characters** (source code) into a sequence of **tokens**.

Characters:

$\underbrace{\text{[a-z]} \quad \text{[A-Z]} \quad \text{[0-9]}}_{\textbf{Alphabets and numbers}} \quad \underbrace{\text{[!,@,\#,\$,\%,\^{},\&,*,(,)[,],.,]}}_{\textbf{Special characters}}$

Tokens:

- **Keywords:** reserved words like **if,else,return,in,while etc..**
- **Identifiers: variable** and **function** names like **x,sum,myFunction etc..**
- **Operators: +,=,-,*,/ ,&,%,!,etc..**
- **Punctuations: like ;,{}[]. etc…**
- **Literals: numeric and string constants like 123,"hello",3.14 etc..**

EXAMPLE:

```
%{
    #include <stdio.h>   //header file
%}
%%                          //lex rule
int|float|return            { printf("KEYWORD: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]*      { printf("IDENTIFIER: %s\n", yytext); }
[0-9]+                      { printf("NUMBER: %s\n", yytext); }
[+\-*/=]                    { printf("OPERATOR: %s\n", yytext); }
.|\n                        { /* Ignore other characters */ }
%%
int main()
{
  yylex();                  // Call lexical analyzer
  return 0;
}
```

YACC (**YET ANOTHER COMPILER COMPILER**)

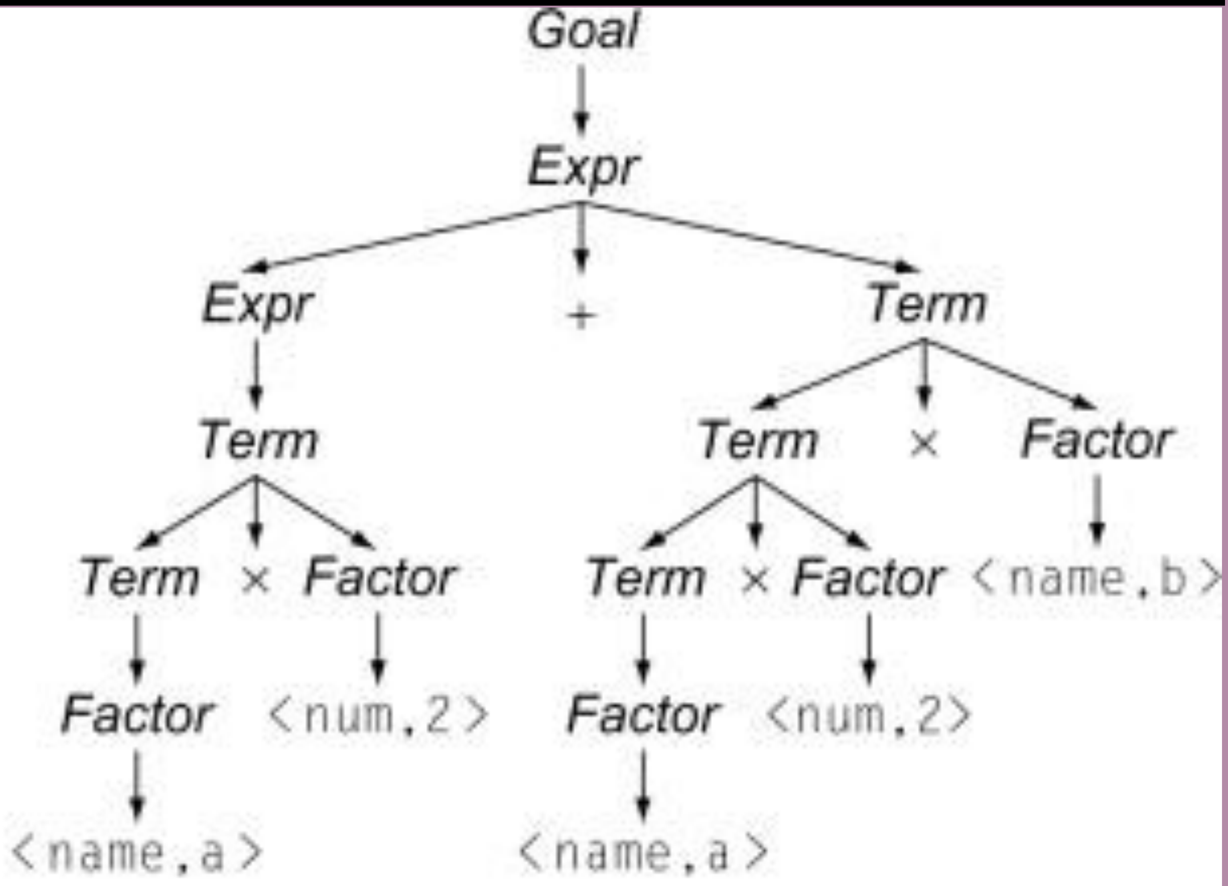
YACC is a **parser generator** used with **Lex** to build compilers or interpreters.

Yacc does:

- Takes **tokens** from Lex.
- Uses a **grammar (BNF - Backus-Naur Form)** to construct a syntax tree.
- Detects and reports **syntax errors.**

Goal	→	*Expr*
Expr	→	*Expr* + *Term*
	|	*Expr* - *Term*
	|	*Term*
Term	→	*Term* × *Factor*
	|	*Term* + *Factor*
	|	*Factor*
Factor	→	<u>(</u> *Expr* <u>)</u>
	|	`num`
	|	`name`

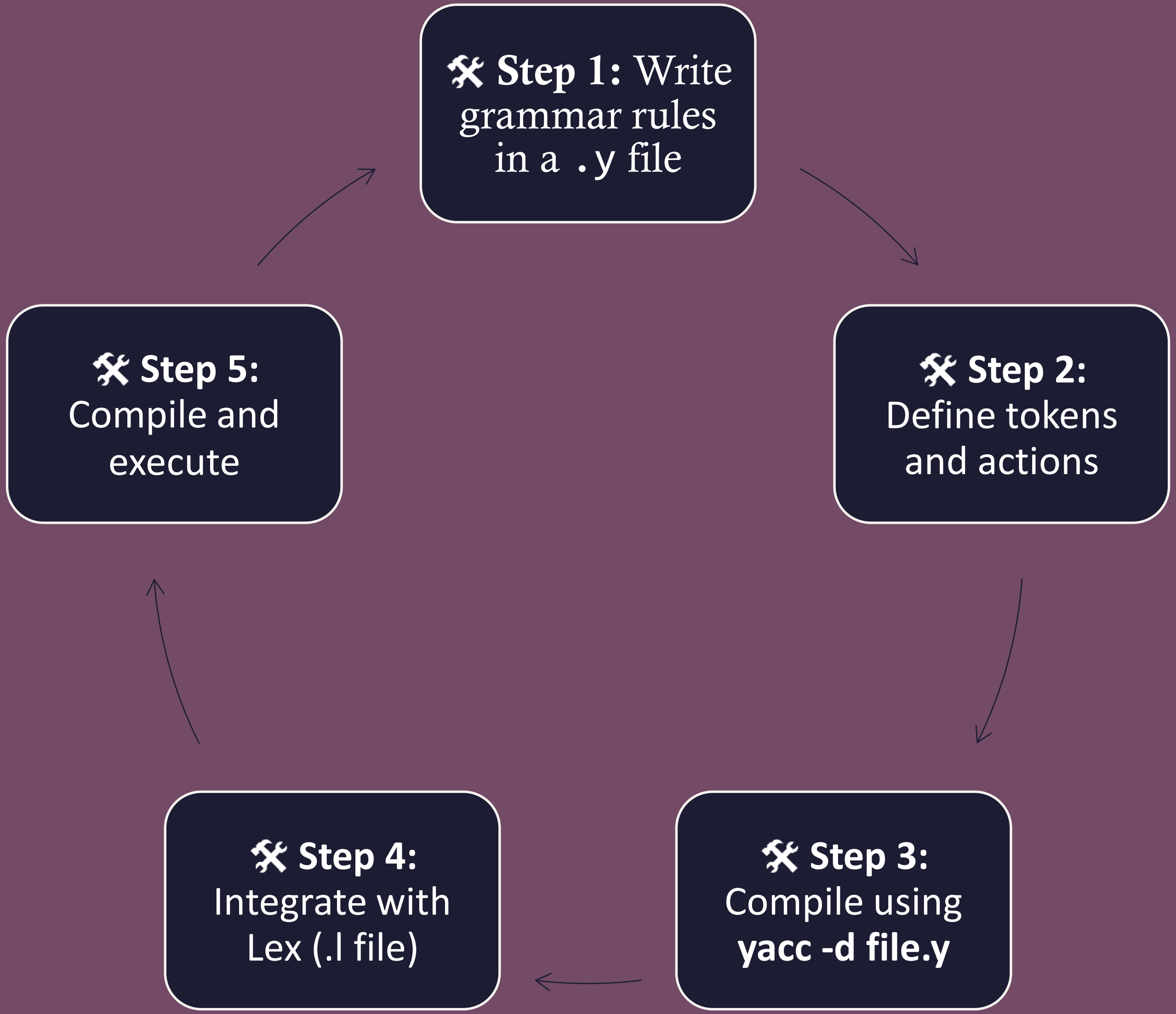
(a) Classic Expression Grammar



(b) Parse Tree for a × 2 + a × 2 × b

Source: <u>ScienceDirect</u>

Yacc workflow



🛠 **Step 1:** Write grammar rules in a .y file

🛠 **Step 2:** Define tokens and actions

🛠 **Step 3:** Compile using **yacc -d file.y**

🛠 **Step 4:** Integrate with Lex (.l file)

🛠 **Step 5:** Compile and execute

```
%{
 #include <stdio.h>   //header file
%}
%token NUMBER PLUS MULTIPLY //tokens defined and passed from lex file
%%
expr: expr PLUS term { printf("Addition\n"); }  //grammar rules-01
          | term
          ;
term: term MULTIPLY factor { printf("Multiplication\n"); } //grammar rules-02
          | factor
          ;
factor: NUMBER  //grammar rule-03
          ;
%%
int yyerror(char *msg) {    //function to handle error
  printf("Syntax Error!\n");
}
int main() {
yyparse();    //to make parsing input as per rules defined.
 return 0;
}
```

THANK YOU....!