YACC

A parser generator is a program that takes as input a specification of a syntax, and produces as output a procedure for recognizing that language. Historically, they are also called compiler-compilers.
YACC (yet another compiler-compiler) is an LALR(1) (Look-Ahead, Left-to-right, Rightmost derivation producer with 1 look-ahead token) parser generator. YACC was originally designed for being complemented by Lex.

Input File:

YACC input file is divided into three parts.

```
/* definitions */
 ....

%%
/* rules */
....
%%

/* auxiliary routines */
....
```

Input File: Definition Part:

- The definition part includes information about the tokens used in the syntax definition:

```
%token NUMBER
%token ID
```

- Yacc automatically assigns numbers for tokens, but it can be overridden by

```
%token NUMBER 621
```

- Yacc also recognizes single characters as tokens. Therefore, assigned token numbers should not overlap ASCII codes.
- The definition part can include C code external to the definition of the parser and variable declarations, within **%{** and **%}** in the first column.
- It can also include the specification of the starting symbol in the grammar:

```
%start nonterminal
```

Input File: Rule Part:

- The rules part contains grammar definition in a modified BNF form.
- Actions is C code in { } and can be embedded inside (Translation schemes).

Input File: Auxiliary Routines Part:

- The auxiliary routines part is only C code.
- It includes function definitions for every function needed in rules part.

- It can also contain the main() function definition if the parser is going to be run as a program.
- The main() function must call the function yyparse().

Input File:

- If yylex() is not defined in the auxiliary routines sections, then it should be included:

```
#include "lex.yy.c"
```

- YACC input file generally finishes with:

```
 .y
```

Output Files:

- The output of YACC is a file named **y.tab.c**
- If it contains the **main()** definition, it must be compiled to be executable.
- Otherwise, the code can be an external function definition for the function **int yyparse()**
- If called with the **–d** option in the command line, Yacc produces as output a header file **y.tab.h** with all its specific definition (particularly important are token definitions to be included, for example, in a Lex input file).
- If called with the **–v** option, Yacc produces as output a file **y.output** containing a textual description of the LALR(1) parsing table used by the parser. This is useful for tracking down how the parser solves conflicts.

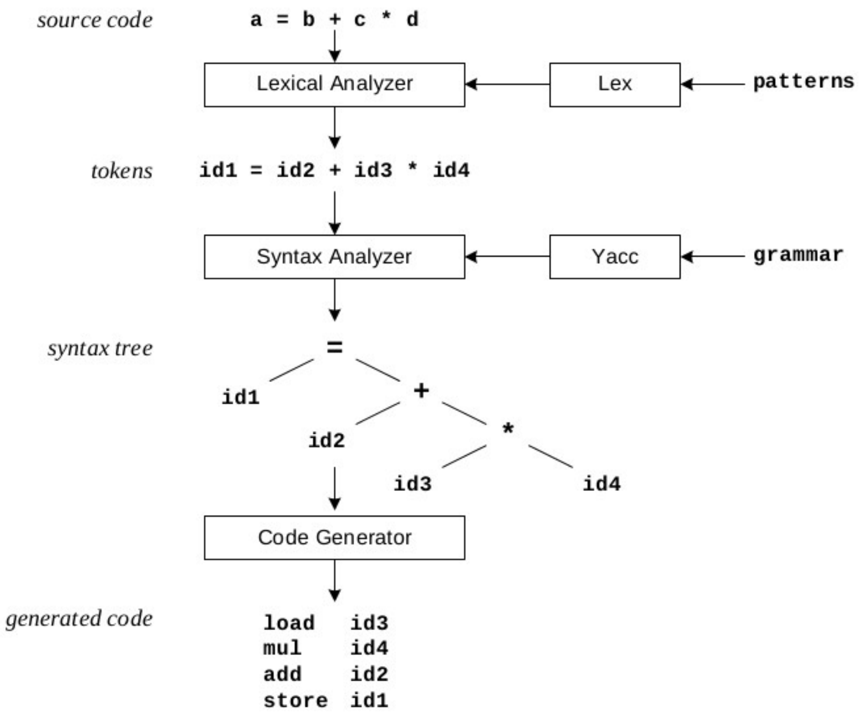

Figure 1: Compilation Sequence

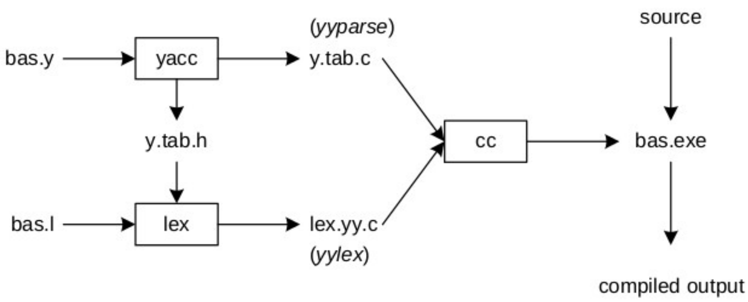

Figure 2: Building a Compiler with Lex/Yacc

Example

Yacc File (.y)

```
%{
      #include <ctype.h>
      #include <stdio.h>
      #define YYSTYPE double /* double type for yacc stack */
%}

%%
      Lines : Lines S '\n'    { printf("OK \n"); }
            | S '\n'
            | error '\n'      {yyerror("Error: reenter last line:");
                               yyerrok; };
      S     : '(' S ')'
            | '[' S ']'
            |   /* empty */   ;
%%

void yyerror(char * s)
/* yacc error handler */
{
      fprintf (stderr, "%s\n", s);
}

int main(void)
{
      return yyparse();
}
```

Lex File (.l)

```
%{
%}

%%
[ \t]       { /* skip blanks and tabs */ }
\n|.        { return yytext[0]; }
%%
```

For Compiling YACC Program:

1. Write lex program in a file file.l and yacc in a file file.y
2. Open Terminal and Navigate to the Directory where you have saved the files.
3. type lex file.l

4. type yacc file.y
5. type cc lex.yy.c y.tab.c -ll -w
6. type ./a.out