YACC

[Yet Another Compiler Compiler]

Prof Sagar Shinde
[Computer Engineering Dept.]
ICEM,Pune

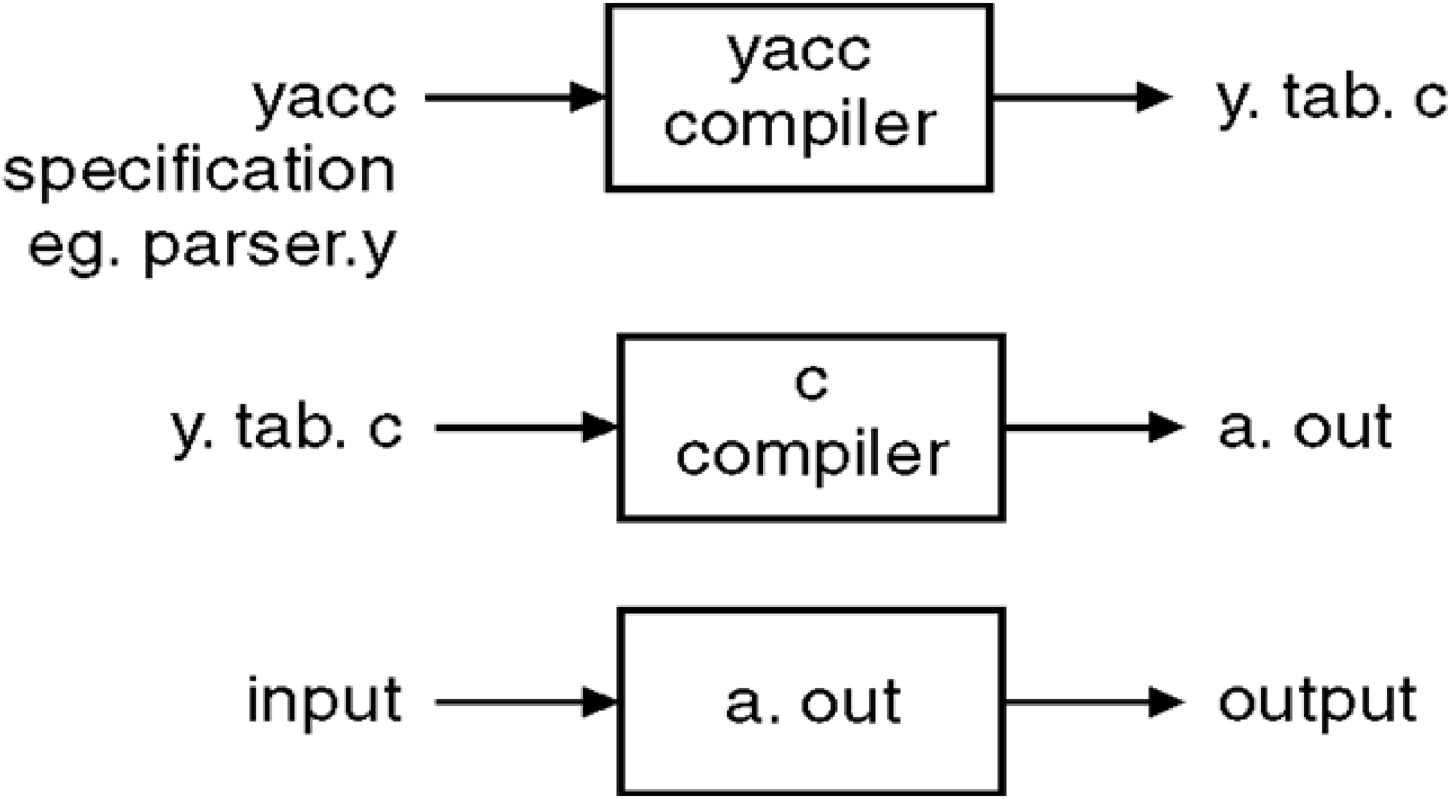
Introduction

- Parser generator facilitates the construction of the front end of a compiler.
- YACC is LALR parser generator.
- It is used to implement hundreds of compilers.
- YACC is command (utility) of the UNIX system.
- YACC stands for “**Y**et **A**nother **C**ompiler **C**omplier”.

YACC Specification:

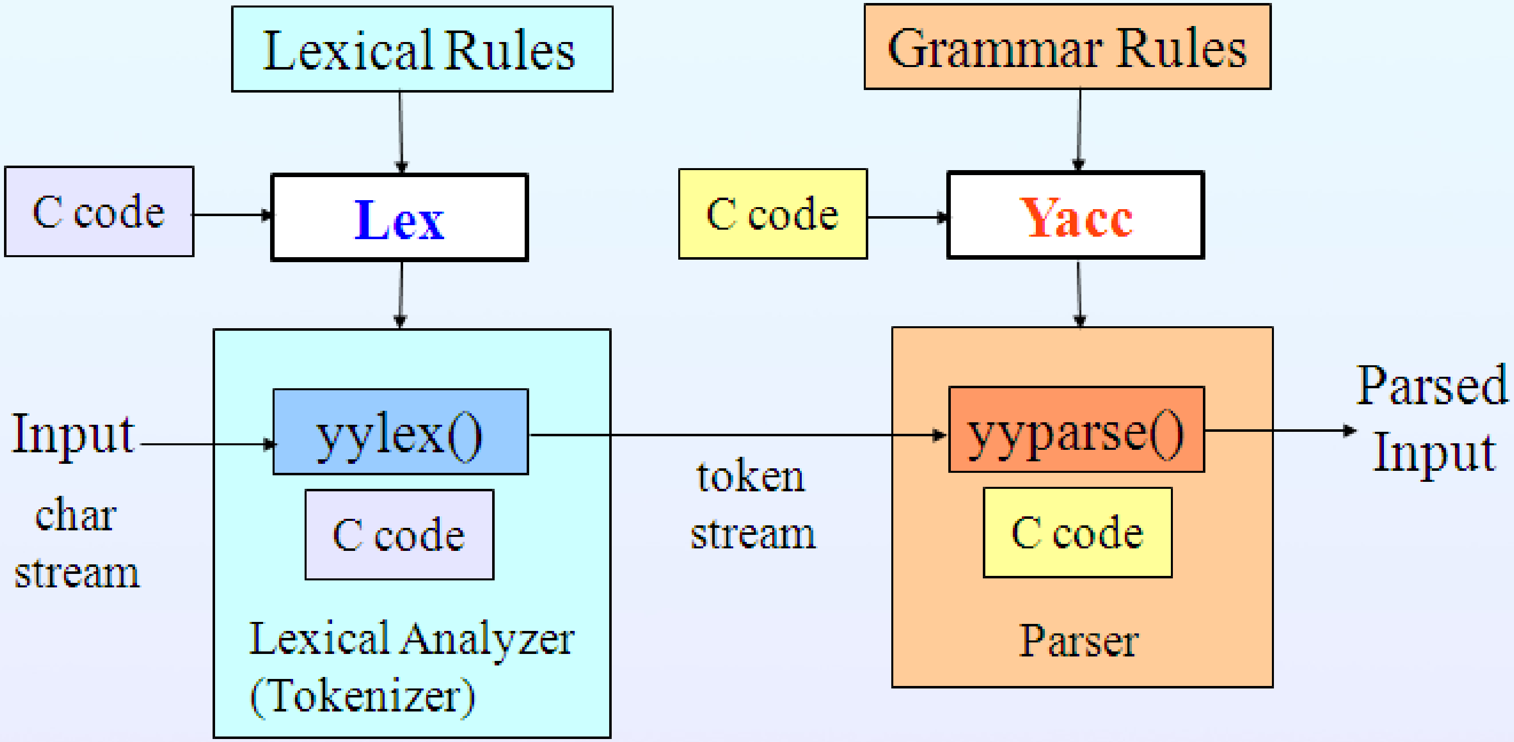
- File in which parser generated is with **·y extension.**
- E.g. parser.y, which is containing YACC specification of the translator. After complete specification UNIX command.
- YACC parser ·y transforms the file parser.y into a C program called y**.**tab**.c** using LR parser.
- The program y**.**tab**.c** is a representation of an LALR parser written in C, along with other C routines that the user may have prepared.
- By compiling y**.**tab**.c** along with the by library that contains the LR parsing program using the command.

  cc y·tab·c – ly
- We obtain the desired object program a.out


yacc specification eg. parser.y
yacc compiler
y. tab. c
y. tab. c
c compiler
a. out
input
a. out
output

Lex and Yacc

- **Lex** and **Yacc** generate C code for your analyzer & parser.



Structure of YACC Program

```
%{
        Definitions Section
%}


%%
        Rules Section (Context Free Grammar)
%%
        Auxiliary Function
```

Definition Section (Declaration) :

- The definitions and programs section are optional.
- Definition section handles control information for the YACC-generated parser and generally set up the execution environment in which the parser will operate.

Declaration Part:

- In declaration section, % { and %} symbol used for C declaration.
- This section is used for definition of token, start, union, associativity and precedence of operator.
- The statement between % { and % } is passed as it is to C program, normally used for comments.

Declaration part: (Contd...)

- **%token NAME NUMBER**

  Used to declare the tokens used in the grammar.

  Eg. % token DIGIT

which declares DIGIT to be token.

- **%start :-**

  Used to declare the start symbol of the grammar.

  Eg.:- %start STMT

- **%type :-**

  Used to create the type of a variable.

  Eg.:- %type <name of any variable> exp

Precedence & Associative

- **%left**

Used to assign the left associatively to operators.

Eg: **%left '+' '-'**

-Assign left associatively to + & – with lowest precedence.

%left '*' '/'

-Assign left associatively to * & / with highest precedence.

- **%right :-**

Used to assign the right associatively to operators.

Eg:- **%right '+' '-'**

- Assign right associatively to + & – with lowest precedence

%right '*' '/'

-Assign right left associatively to * & / with highest precedence.

Precedence & Associative

- **%nonassoc :-**

  Used to unary associate.

Eg.:- %nonassoc UMINUS

- **%prec :-**

  Used to tell parser use the precedence of given code.

Eg.:- %prec UMINUS

Rule Section:

- In YACC specification after the first %% pair, we put the translation rules.
- Each rule consists of a grammar production and the associated semantic action.
- It means that YACC rules define what is a legal sequence of tokens in our specifications language.

Contd…

- A set of productions (CFG) of form

<left side> → <alt 1> | <alt 2> | … <alt n>

can be written in YACC as

```
<left side>   :     <alt 1>     {action 1}
              |     <alt 2>     {action 2}
              …
              …
              |     <alt n>     {action n}
              ;
```

Contd…

- A YACC semantic action is a sequence of statements.
- In a semantic action, the symbol $$ refers to the attribute value associated with the non-terminal of the left while $i refer to the value associated with the ith grammar symbol.

E.g. The two E-productions

E → E + T/T in YACC

```
Exp   : Exp '+' Term          {$$ = $1 + $3;}
        | Term
         ;
```

Contd…

- In above production:

  exp is $1,

  ‘+’ is $2 and

  term is $3.

- The semantic action associated with first production adds values of exp and term and result of addition copying in $$ (exp) left hand side.
- For above second number production, we have omitted the semantic action since it is just copying the value.
- {$$ = $1;} is the default semantic action.

Token types :

- Token data types are declared in YACC using the YACC declaration % union, like this :

```
% union
     {
            char * str ;
             int num ;
      }
```

Token types : (Contd…..)

- This variable data type is required when token is holding some value and we have to specify which kind of value it is holding.
- Normally yylval is being defined a union as types (char*) and int.
- We use this variable declaration to specify the type which is associated with token in following manner :

% token <str> EXE

% token <num> DIGIT

- Note that we have the token value, we want to use it. This value is used by YACC in above maintained variables.

e.g. $$, $1 …etc.

- This token declaration is in YACC declaration section.

% type <num> default

- This is same approach as we used for % token definitions but this is not used by the lex.

Subroutines :

- YACC generates a single function called yyparse().
- This function requires no parameters and returns either a 0 on success, and 1 on failure. If syntax error over its return 1.
- The special function yyerror() is called when YACC encounters an invalid syntax.
- The yyerror() is passed a single string (char∗) argument. This function just prints “parse error” message, it is possible to give your own message in this function like

```
    yyerror (char ∗error)
      {
            fprintf (stderr, “% S \ n”, error);
       }
```

Subroutines (Contd…)

- When LEX and YACC work together lexical analyzer yylex () to produce pairs consisting of a token and its associated attribute value.
- If a token such as DIGIT is returned, the token value associated with a token is communicated to the parser through a YACC defined variable yylval.
- We have to return tokens from lex to YACC, where its declaration is in YACC. To link this lex program include a y.tab.h file, which is generated after YACC the program.