

Parser Generators: YACC

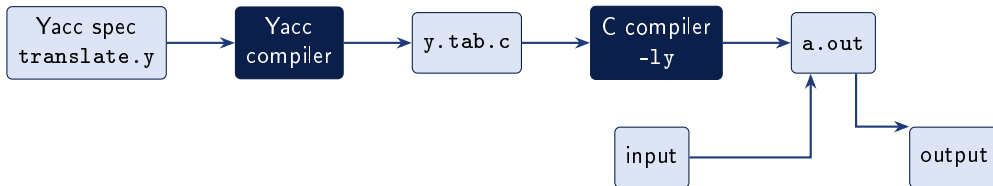
Compiler Construction, Lecture 10

K.R. Chowdhary, Professor of CS

August 19, 2019 | Fall 2019

Parser Generators and the YACC Workflow

A parser generator builds the front end of a compiler from a grammar. **YACC** (“Yet Another Compiler Compiler”) is an LALR(1) parser generator, created by S. C. Johnson at Bell Labs / AT&T and available on UNIX and Linux.



Commands

```
$ yacc translate.y           # produces y.tab.c (LALR parser in C)
$ cc y.tab.c -ly             # link with the ly library (LR parsing program)
$ ./a.out                    # run the translator
```

Structure of a Yacc Specification

Three parts

declarations

%%

translation rules

%%

supporting C routines

Rule form:

```
<head> : <body>1 {action1}  
        | <body>2 {action2}  
        ;
```

- **Declarations** (both optional): C declarations between %{ and %}, and token declarations such as %token DIGIT. If Lex is used, the token declarations are shared with the scanner
- **Rules**: unquoted names that are not tokens are non-terminals; a quoted character like 'c' is a terminal. The first head is the start symbol
- **Actions** are C statements run when the production is reduced: \$\$ is the head's attribute value, \$i the value of the *i*th body symbol. Default action: \$\$ = \$1;
- **Supporting routines**: a lexical analyser yylex(); token attributes are passed through yylval

Example 10.1: Desk Calculator (deskcal.y)

Declarations and rules

```
%{
#include <ctype.h>
}%
%token DIGIT
%%
line   : expr '\n'      { printf("%d\n", $1); } ;
expr   : expr '+' term  { $$ = $1 + $3; }
        | term ;
term   : term '*' factor { $$ = $1 * $3; }
        | factor ;
factor : '(' expr ')'   { $$ = $2; }
        | DIGIT ;
%%
```

Grammar: $E \rightarrow E + T \mid T$,
 $T \rightarrow T * F \mid F$, $F \rightarrow (E) \mid \textit{digit}$,
with single-digit tokens.

- `line` is the added start production: an expression followed by a newline; its action prints the value
- In `expr '+' term`, `term` is the third body symbol and `'+'` the second, so the action adds `$1` and `$3`
- Single-symbol bodies use the default action

Supporting C routine

```
yylex() {  
    int c = getchar();  
    if (isdigit(c)) {  
        yylval = c - '0';  
        return DIGIT;  
    }  
    return c;  
}
```

- A digit stores its value in `yylval` and returns the token `DIGIT`
- Any other character is returned as its own token name; the parser calls `yylex()` for each token
- Input digits are single digits; the result is printed with `%d`, so `3*4` prints `12`

Build and run

```
$ yacc deskcal.y  
$ cc y.tab.c -ly  
$ ./a.out  
2+3  
5
```

Review questions

1. What does YACC stand for?
2. Does YACC produce top-down or bottom-up parsers? What grammar class does it accept?
3. What is a Yacc semantic action made of?
4. What is the input of Lex?

Exercises

1. Which are the semantic actions in Example 10.1? Explain each
2. Extend the calculator for subtraction and division
3. Run $2+3$, $2*3$, $2+(2+2)$, $2*(2+3)$, $3*4$ and justify each result

Reference: S. C. Johnson, *Yacc: Yet Another Compiler-Compiler*, AT&T Bell Laboratories, 1975.