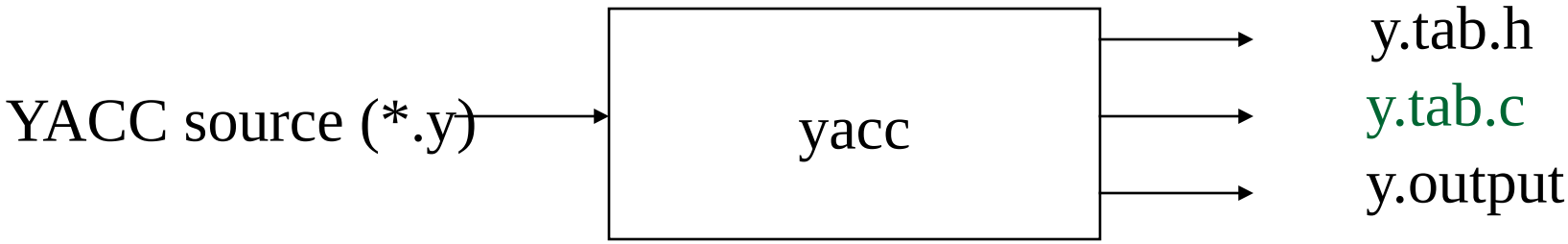
YACC Parser Generator

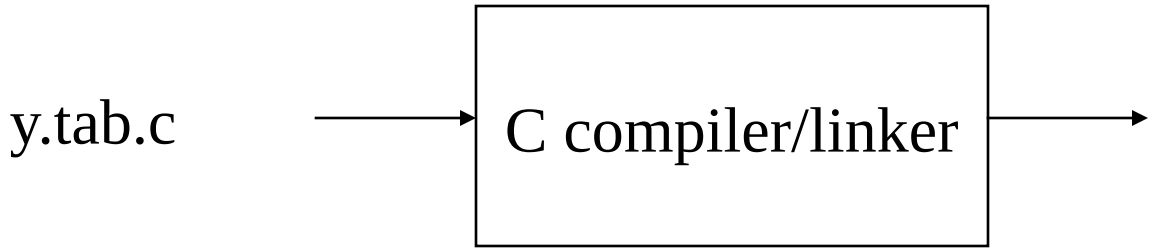
YACC

- **YACC (Yet Another Compiler Compiler)**
- **Produce a parser for a given grammar.**
  - Compile a LALR(1) grammar
- **Original written by *Stephen C. Johnson*, 1975.**
- **Variants:**
  - lex, yacc (AT&T)
  - bison: a yacc replacement (GNU)
  - flex: *f*ast *lex*ical analyzer (GNU)
  - BSD yacc
  - PCLEX, PCYACC (Abraxas Software)

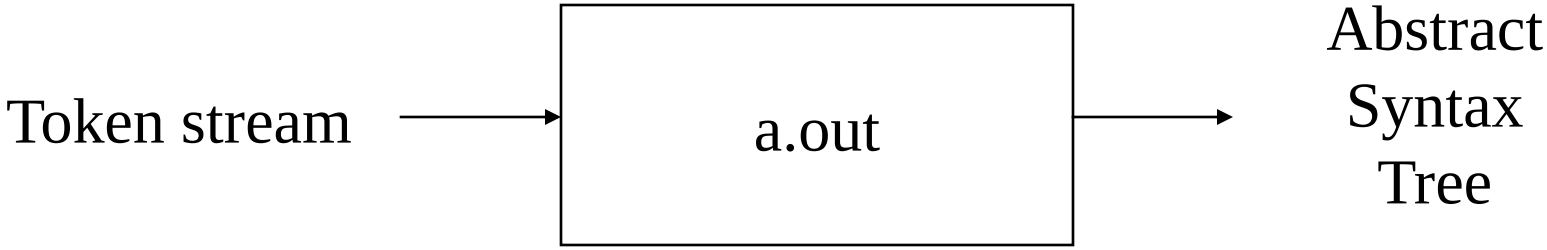
How Does YACC Work?



Generate a new parser code from grammar



Compile a new parser



Parse source code

YACC Format

```
%{
    C declarations
%}
    yacc declarations
%%
    Grammar rules
%%
    Additional C code
```

YACC Example

C declarations

yacc declarations

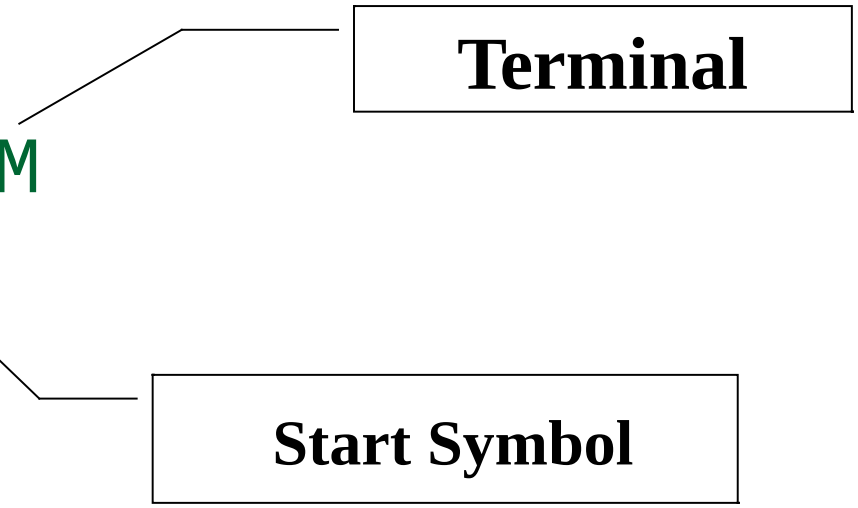
Grammar rules

Additional C code

```
%{
#include <stdio.h>
%}

%token NAME NUMBER
%%

statement: NAME '=' expression
        | expression            { printf("= %d\n", $1); }
        ;

expression: expression '+' NUMBER { $$ = $1 + $3; }
        |   expression '-' NUMBER { $$ = $1 - $3; }
        |   NUMBER                { $$ = $1; }
        ;
%%
int yyerror(char *s)
{
   fprintf(stderr, "%s\n", s);
   return 0;
}

int main(void)
{
   yyparse();
   return 0;
}
```

YACC Declarations Section

```
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token ID NUM
%start expr
```



YACC Declaration Summary

`%start'

Specify the grammar's start symbol

`%union'

Declare the collection of data types that semantic values may have

`%token'

Declare a terminal symbol (token type name) with no precedence or associativity specified

`%type'

Declare the type of semantic values for a nonterminal symbol

YACC Declaration Summary

`%right'

Declare a terminal symbol (token type name) that is right-associative

`%left'

Declare a terminal symbol (token type name) that is left-associative

`%nonassoc'

Declare a terminal symbol (token type name) that is nonassociative

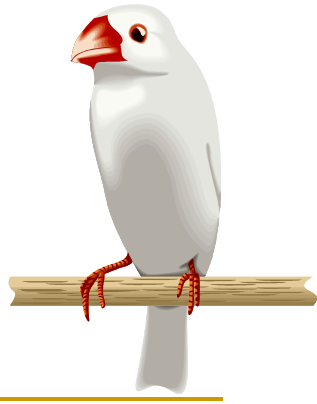
(using it in a way that would be associative is a syntax error,

ex: x *op*. y *op*. z is syntax error)

Grammar Rules Section

- **Normally written like this**
- **Example:**

```
expr   : expr '+' term
       | term
       ;
term   : term '*' factor
       | factor
       ;
factor : '(' expr ')'
       | ID
       | NUM
       ;
```



Work between LEX and YACC

- **Use enumeration / define**
  - YACC produce y.tab.h
  - LEX include y.tab.h
  - yacc -d gram.y
    - Will produce y.tab.h

YACC and Bison Command

- **Yacc (AT&T)**
  - yacc –d *xxx.y*
- **Bison (GNU)**
  - bison –d **–y** *xxx.y*

Produce y.tab.c, the same as above yacc command

Work between LEX and YACC

scanner.l

```
%{
#include <stdio.h>
#include "y.tab.h"
%}
id          [_a-zA-Z][_a-zA-Z0-9]*
%%
int         { return INT; }
char        { return CHAR; }
float       { return FLOAT; }
{id}        { return ID;}
```

parser.y

```
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token  CHAR, FLOAT, ID, INT
%%
```

```
yacc -d xxx.y
```

- **Produced**

```
y.tab.h

# define CHAR 258
# define FLOAT 259
# define ID 260
# define INT 261
```

Debug YACC Parser

1. **Use –t option** or **define YYDEBUG to 1.**
2. **Set variable *yydebug* to 1 when you want to trace parsing status.**
3. **If you want to trace the semantic values**
   - Define your **YYPRINT** function

```
#define YYPRINT(file, type, value) yyprint(file, type, value)

static void
yyprint (FILE *file, int type, YYSTYPE value)
{
  if (type == VAR)
    fprintf (file, " %s", value.tptr->name);
  else if (type == NUM)
    fprintf (file, " %d", value.val);
}
```

Simple Calculator Example - Files

- **calc.l**
  - Specifies the lex command specification file that defines the lexical analysis rules.
- **calc.y**
  - Specifies the yacc command grammar file that defines the parsing rules, and calls the yylex subroutine created by the lex command to provide input.

Simple Calculator Example – YACC File

- %{
- #include <stdio.h>
- int regs[26];
- int base;
- %}
- %start list
- %token DIGIT LETTER
- %left '|'
- %left '&'
- %left '+' '-'
- %left '*' '/' '%'
- %left UMINUS /*supplies precedence for unary minus */

Simple Calculator Example – YACC File

```
%%                  /* beginning of rules section */
list:                 /*empty */
        |
        list stat '\n'
        |
        list error '\n'
        {
          yyerrok;
        }
        ;
stat:    expr
        {
          printf("%d\n",$1);
        }
        |
        LETTER '=' expr
        {
          regs[$1] = $3;
        }
        ;
```

```
expr:    '(' expr ')'
        {
          $$ = $2;
        }
        |
        expr '*' expr
        {
          $$ = $1 * $3;
        }
        |
        expr '/' expr
        {
          $$ = $1 / $3;
        }
        |
        expr '%' expr
        {
          $$ = $1 % $3;
        }
        |
        expr '+' expr
        {
          $$ = $1 + $3;
        }
        |
        expr '-' expr
        {
          $$ = $1 - $3;
        }
        |
        expr '&' expr
        {
          $$ = $1 & $3;
        }
        |
        expr '|' expr
        {
          $$ = $1 | $3;
        }
        |
```

Simple Calculator Example – YACC File

```
        '-' expr %prec UMINUS
        {
          $$ = -$2;
        }
        |
        LETTER
        {
          $$ = regs[$1];
        }
        |
        number
        ;
    number:  DIGIT
        {
          $$ = $1;
          base = ($1==0) ? 8 : 10;
        }       |
        number DIGIT
        {
          $$ = base * $1 + $2;
        }
        ;
```

Simple Calculator Example – YACC File

```
%%
main()
{
 return(yyparse());
}
yyerror(s)
char *s;
{
  fprintf(stderr, "%s\n",s);
}
yywrap()
{
  return(1);
}
```

Simple Calculator Example – Lex File

```
%{

#include <stdio.h>
#include "y.tab.h"
int c;
extern int yylval;
%}
%%
" "       ;
[a-z]     {
            c = yytext[0];
            yylval = c - 'a';
            return(LETTER);
          }
[0-9]     {
            c = yytext[0];
            yylval = c - '0';
            return(DIGIT);
          }
[^a-z0-9\b]    {
                 c = yytext[0];
                 return(c);
               }
```

Simple Calculator Example – Compile and Run

- **bison -d –y calc.y**
  - create y.tab.c and y.tab.h
- **flex calc.l**
  - create lex.yy.c
- **gcc -g lex.yy.c y.tab.c -o calc**
  - Create execution file
- **./calc**
  - Run the calculator