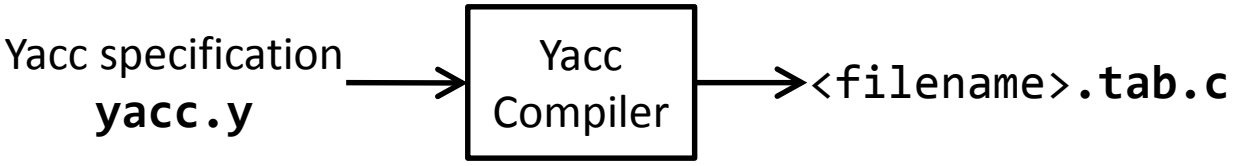
Yet Another Compiler Compiler (Yacc)

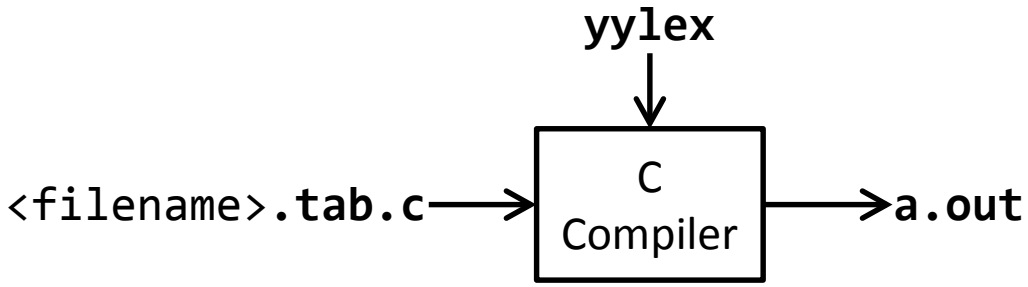
An LALR(1) Parser Generator

What is Yacc

- Yet another compiler compiler
- Parser generator
  - Compiler-compilers
  - **Bison** is the most popular GNU implementation



- Input
  - A specification file
    - ✓ `<filename>.y`
  - **`yylex`** procedure
    - ✓ Returns the next lookahead token



- Output
  - C code for the parser
    - ✓ `<filename>.tab.c` or `y.tab.c` or `ytab.c` or

Expression Grammar

{ definitions }

%%

{ rules }

%%

{ auxiliary routines }

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

%token     PLUS MINUS TIMES LPAR RPAR NUMBER CR OTHER
%start     input

%%

input      :
           | input line
           ;
line       : CR
           | exp CR              { printf("%d\n", $1); }
           ;
exp        : exp PLUS term       { $$ = $1 + $3; }
           | exp MINUS term      { $$ = $1 - $3; }
           | term                { $$ = $1; }
           ;
term       : term TIMES factor   { $$ = $1 * $3; }
           | factor              { $$ = $1; }
           ;
factor     : NUMBER              { $$ = $1; }
           | LPAR exp RPAR       { $$ = $2; }
           ;
%%
main() { return yyparse(); }
int yyerror(char *s) { fprintf(stderr, "%s\n", s); return 0; }
```

Lexical Analyser

```
%{
#include "exp.tab.h"
%}

DIGIT        [0-9]

%%

[0-9]+       { yylval = atoi(yytext); return NUMBER; }
"+"          { return PLUS; }
"-"          { return MINUS; }
"*"          { return TIMES; }
"("          { return LPAR; }
")"          { return RPAR; }
"\n"         { return CR; }
.            { return OTHER; }

%%
```

Exercise

- **`bison -d <filename>.y`**
  - Produces the **`<filename>.tab.c`** file
  - **`-d`** flag produces the **`<filename>.tab.h`** file containing the token declarations
  - **`-v`** verbose option produces a **`<filename>.output`** file with the LALR(1) parsing table
  - **`--debug`** flag traces the execution of the parser
    - ✓ Defines the symbol **YYDEBUG** to be 1

- **`lex <filename>.l`**
  - Must include the **`<filename>.tab.h`** file
  - Produces the **`lex.yy.c`** file

- **`gcc -o <filename> <filename>.tab.c lex.yy.c -lfl`**

Important Features

- Two ways of recognising tokens
  - **`%token token_list`**
    - ✓ Provided by the **`yylex`** function
  - Single character tokens can be included directly in grammar rules (e.g. **'+', '-', '*'**)

- Start symbol
  - Nonterminal listed first
  - **`%start symbol`**

- Pseudo-variables
  - **$$** represents the nonterminal being recognised
  - **$1**, **$2**, **$3**, etc. represent values of each symbol in the right-hand side

- Tokens may be assigned values during scanning in the **`yylval`** variable

Yacc internal name / Definition mechanism	Meaning / Use
`y.tab.c`	Output file name
`y.tab.h`	Yacc-generated header file containing token definitions
`yyparse`	Yacc parsing routine
`yylval`	Value of current token in stack
`yyerror`	User-defined error message printer used by Yacc
`error`	Yacc error pseudotoken
`yyerrok`	Procedure that resets parser after error
`yychar`	Contains the lookahead token that caused an error
`YYSTYPE`	Preprocessor symbol that defines the value type of the parsing stack
`yydebug`	Variable which, if set by the user to 1, causes the generation of runtime information on parsing actions
`%token`	Defines token preprocessor symbols
`%start`	Defines the start nonterminal symbol
`%union`	Defines a union **YYSTYPE**, allowing values of different types on parsing stack
`%type`	Defines the variant union type returned by a symbol
`%left %right %nonassoc`	Defines the associativity and precedence (by position) of operators

Value Types

```
%union { double val;
         char op; }

%type   <val>   exp term factor
%type   <op>    addop mulop

exp     : exp op term  {       switch($2) {
                               case '+':      $$ = $1 + $3;  break;
                               case '-':      $$ = $1 + $3;  break;
                               }
                     }
        | term { $$ = $1; }
        ;

op      : '+'   { $$ = '+'; }
        | '-'   { $$ = '-'; }
        ;
```

- Defined by the preprocessor symbol **YYSTYPE**

Parsing Conflicts


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

%token   NUMBER

%%

exp      : exp ‘+’ exp     { $$ = $1 + $3; }
         | NUMBER          { $$ = $1; }
         ;

%%

main() {
   return yyparse();
}

int yyerror(char *s) {
  fprintf(stderr, "%s\n", s);
  return 0;
}
```

```
bison –v exp.y
exp.y: conflicts: 1 shift/reduce
state 5

1 exp: exp . PLUS exp
1    | exp PLUS exp .

PLUS  shift, and go to state 4

PLUS      [reduce using rule 1 (exp)]
 $default  reduce using rule 1 (exp)
```

Disambiguating Rules

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

%left               '+' '-'
%left               '*'

%%

exp       : exp '+' exp       { $$ = $1 + $3; }
          | exp '-' exp       { $$ = $1 - $3; }
          | exp '*' exp       { $$ = $1 * $3; }
          | '(' exp ')'       { $$ = $2; }
          | NUMBER            { $$ = $1; }
          ;

%%

main() {
    return yyparse();
}

int yyerror(char *s) {
  fprintf(stderr, "%s\n", s);
  return 0;
}
```

- **+** and **–** have the same precedence and are left associative
- ***** is left associative and has higher precedence
- Other possible Yacc operator specifications are **%right** and **%nonassoc**

Embedded Actions

```
int current_type;

%%

decl    :       type            { current_type = $1; }
                var_list
        ;
type    :       INT             { $$ = INT_TYPE; }
        |       FLOAT           { $$= FLOAT_TYPE; }
        ;
var_list: var_list ‘,’ ID       { setType(tokenString, current_type); }
        | ID                    { setType(tokenString, current_type); }
```

Error Handling

```
%{
#include <stdio.h>
%}
%token      PLUS MINUS TIMES LPAR RPAR NUMBER CR OTHER
%start      input

%%

input                   :
            | input line
            ;
line        : CR
            | exp CR                { printf("%d\n", $1); }
            | error CR              { yyerror("incorrect expression"); }
            ;
exp         : exp PLUS term         { $$ = $1 + $3; }
            | exp MINUS term        { $$ = $1 - $3; }
            | term                  { $$ = $1; }
            ;
term        : term TIMES factor     { $$ = $1 * $3; }
            | factor                { $$ = $1; }
            ;
factor      : NUMBER                { $$ = $1; }
            | LPAR exp RPAR         { $$ = $2; }
            ;
%%
main() { return yyparse(); }
int yyerror(char *s) { fprintf(stderr, "%s\n", s); return 0; }
```