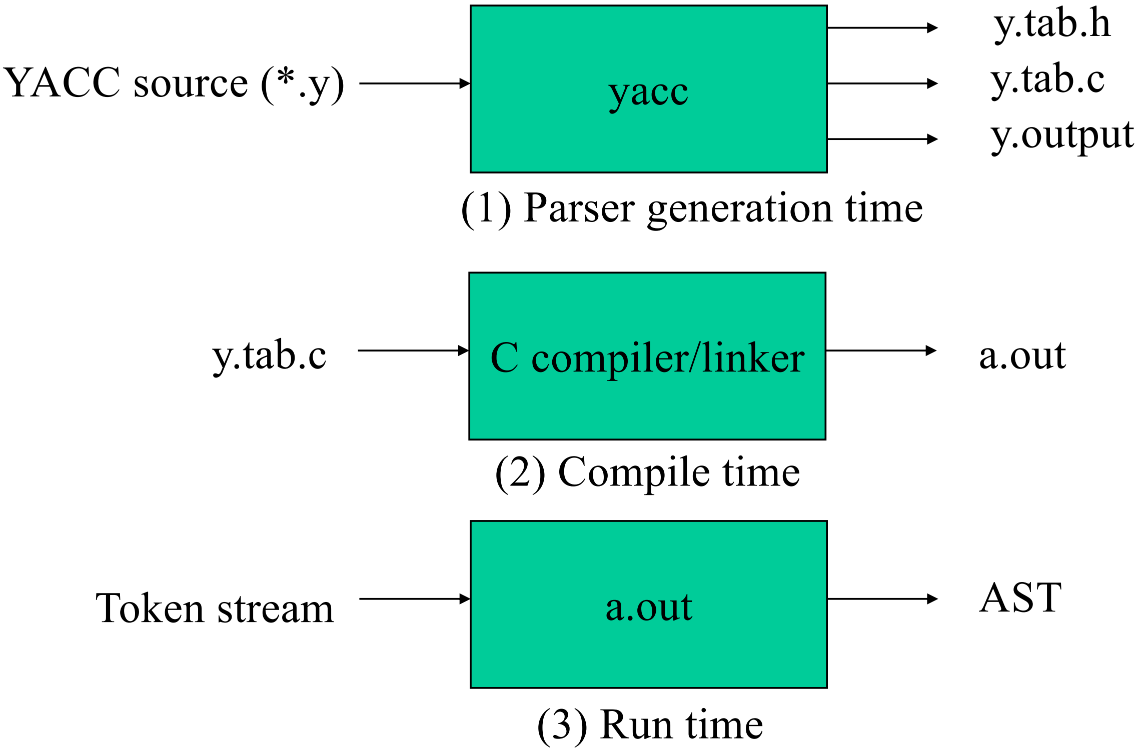
Lecture on YACC
(Yet Another Compiler-compiler)

Introduction

- YACC (Yet Another Compiler Compiler) is a program designed to compile a LALR(1) grammar and to produce the source code of the syntactic analyzer of the language produced by this grammar.
- It is also possible to perform semantic actions.
- Written by Stephen C. Johnson, 1975.
- Variants: YACC(AT&T), BISON (GNU), PCYACC.

How YACC Works



YACC source (*.y)

yacc

y.tab.h

y.tab.c

y.output

(1) Parser generation time

y.tab.c

C compiler/linker

a.out

(2) Compile time

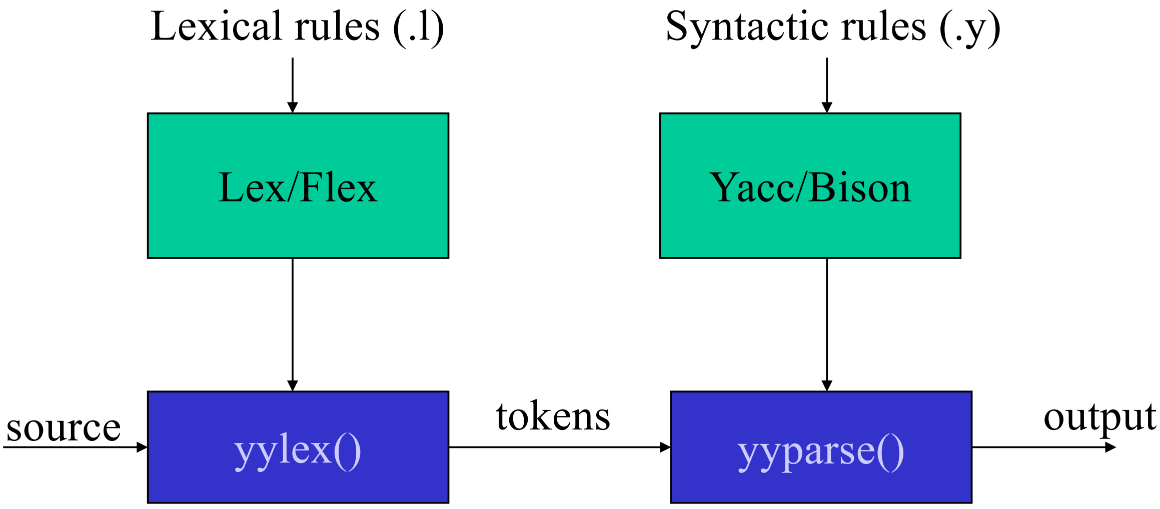
Token stream

a.out

AST

(3) Run time

Works with Lex



Lexical rules (.l)

Syntactic rules (.y)

Lex/Flex

Yacc/Bison

source

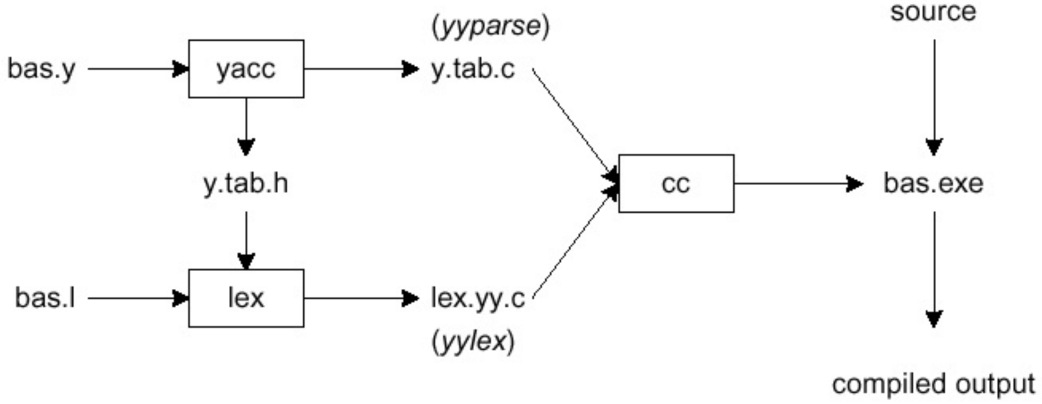
yylex()

tokens

yyparse()

output

Building a Compiler With Lex/Yacc



Bottom-Up
Reverse rightmost

```
1     E -> E + E
2     E -> E * E
3     E -> id
```

```
 1    . x + y * z      shift
 2    x . + y * z      reduce(r3)
 3    E . + y * z      shift
 4    E + . y * z      shift
 5    E + y . * z      reduce(r3)
 6    E + E . * z      shift
 7    E + E * . z      shift
 8    E + E * z .      reduce(r3)
 9    E + E * E .      reduce(r2)
10    E + E .          reduce(r1)
11    E .              accept
```

Structure of a YACC Program

%{

C declarations

%}

yacc declarations

%%

Grammar rules

%%

Additional C code

– only the first %% and the second part are mandatory

Declaration Part

- Specifications written in target language (C), enclosed between %{ and %}

```
 %{
#define  YYSTYPE TreeNode *
#include   “util.h”
static char * savedName; /* for use in assignments */
…
%}
```

- Declaration of the tokens

```
%token IF THEN ELSE END REPEAT READ WRITE
%token ID NUM
```

Declaration Part

- Operators' priority or associativity
- The *type* of the terminal, using the reserved word “%union”: (*typed token*)

```
%union {
  double dval;
  int vblno;
}
%token <vblno> NAME
%token <dval> NUMBER
%left '-' '+’
%left '*' '/’
%nonassoc UMINUS
```

UMINUS has the highest precedence

Production Part

- This part is a specification of the grammar in LALR(1) of whatever we want to parse.
- If the grammar is ambiguous, you will get error messages such as shift/reduce conflicts and/or reduce/reduce conflicts. • May include semantic action.

```
%start stmts    /* or default to the first nonterminal*/
%%
stmts: stmts stmt
      |   ;
stmt: assignment | if_stmt | ...
…
%%
```

Production Part

- To obtain the values returned by previous actions and the lexical analyzer, the action can use the pseudo-variables $1, $2, …, $n
- The pseudo-variable $$ represents the value returned by the complete action

Actions

```
expr:     expr ‘+’ expr      {$$ = $1 + $3;}
       |  expr ‘*’ expr      {$$ = $1 * $3;}
       |  ‘(‘ expr ‘)’       {$$ = $2;}

expr:     expr ‘+’ expr      {$$ = makenode(’+’, $1, $3);}
       |  expr ‘*’ expr      {$$ = makenode(‘*’, $1, $3);}
       |  ‘(‘ expr ‘)’       {$$ = $2;}
```


Support Code Part

- This optional section may contain a number of supporting C functions or compiler directives to include a file containing these functions.
- The parser also requires that a scanner yylex() be provided.

```
%%
void yyerror(char *)
{ … }
void main(void) {
    yyparse();
}
```

- The function yyerror() allows user to specify action taken by the parser when a finite state machine enters an error state.

Example: A small calculator

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
%}

%%
[0-9]+    {
            yylval = atoi(yytext);
            return NUMBER;
          }
\n        return 0;
[ \t]     ;
.         return yytext[0];
```

lex file: d.l

```
#ifndef YYSTYPE
#define YYSTYPE int
#endif
#define NAME    257
#define NUMBER  258


extern YYSTYPE yylval;
```

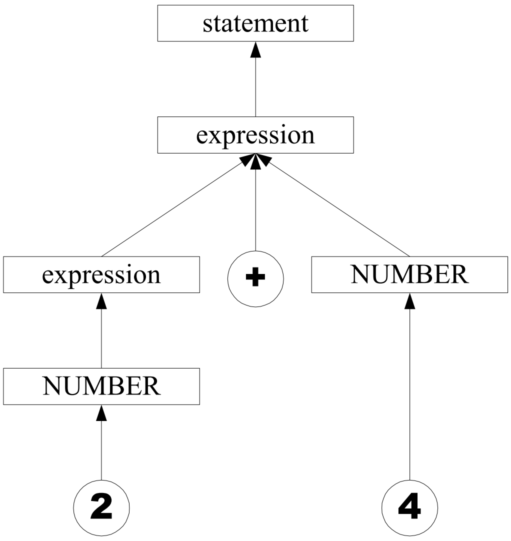
y.tab.h

```
%{
#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 file: d.y

```
% bison -y -d d.y
% flex d.l
% gcc y.tab.c lex.yy.c -ll -ly
% ./a.out
2+4
= 6
%./a.out
10+-4
syntax error
%
```

```
statement => expression
          => expression + NUMBER
          => expression + 4
          => NUMBER + 4
          => 2 + 4
```



Communication between Lex and YACC

- Lex predefined variables
  - yytext: pointer to matched string.
- YACC
  - yylval: value (attribute) of token.

Token/Non-terminal Value Types

- The declaration

```
%union {
      double dval;
      int vblno;
}
```

is translated to

```
%typedef  union {
                double dval;
                int vblno;
              }  YYSTYPE;
```

- Structured values are also allowed.

```
#define YYSTYPE  TreeNode *
{ $$.left = $1.right; }
```

Example Refined

```
…
%token <val_value> NUMBER
%token <val_number> NAME
%%
statement_list:    statement '\n'
            |      statement_list statement '\n'
            ;
statement:         NAME '=' expression          { vbltable[$1] = $3; }
            |      expression                   { printf("= %g\n", $1); }
            ;
expression:        expression '+' expression    { $$ = $1 + $3; }
            |      expression '-' expression    { $$ = $1 - $3; }
            |      expression '*' expression    { $$ = $1 * $3; }
            |      expression '/' expression    { if($3 == 0) yyerror("divide by zero");
                                                  else $$ = $1 / $3;}
            |      '-' expression  %prec UMINUS          { $$ = -$2; }
            |      '(' expression ')'  ‘                 { $$ = $2; }
            |      NUMBER
            |      NAME                                  { $$ = vbltable[$1]; }
            ;
```

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
%}

%%
[0-9]+          {
                   yylval.var_value = atoi(yytext);
                   return NUMBER;
                }
[a-z]           {
                   yylval.var_number = yytext[0] - 'a';
                   return NAME;
                }
"$"             return 0;
[ \t]           ;
\n |
.               return yytext[0];
```

```
% ./a.out
a=100
b=20
a=a+b-10
a
= 110
abc=10
= 110
parse error
```

Embedded Actions (Mid-Rule Action)

- Occasionally it is necessary to execute some code prior to the complete parsing of a grammar rule.
- A mid-rule action may refer to the components preceding it using $*n*, but it may not refer to subsequent components because it is run before they are parsed.
- The mid-rule action itself counts as one of the components of the rule. (i.e. has semantic value)
- Ex: A: B { /* Embedded action)/ } C ;

An Example of Embedded Action

- *assignment* statement

```
assign_stmt :   ID
                { savedName = copyString(yytext);
                  savedLineNo = lineno; }
                ASSIGN  exp
                { $$ = newStmtNode(AssignK);
                  $$->child[0] = $4;
                  $$->attr.name = savedName;
                  $$->lineno = savedLineNo;
                }
```

Conflicts

- Shift/Reduce conflict
  Default resolution: Shift

- Reduce/Reduce conflict
  Default resolution: Reduce the rule declared earlier

- When there are more than one operator appear in a single rule, YACC uses the precedence of the rightmost operator's as the precedence of the rule

Error Messages

- Bad error message:
  - – Syntax error.
- It is better to track the line number in lex:

```
void yyerror(char *s)
{
   fprintf(stderr, "line %d: %s\n:", yylineno, s);
}
```

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)