中山大學 SUN YAT-SEN UNIVERSITY

国家超级计算广州中心 NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

Compiler Design
编 译 器 构 造 实 验

Lab 4：YACC

张献伟

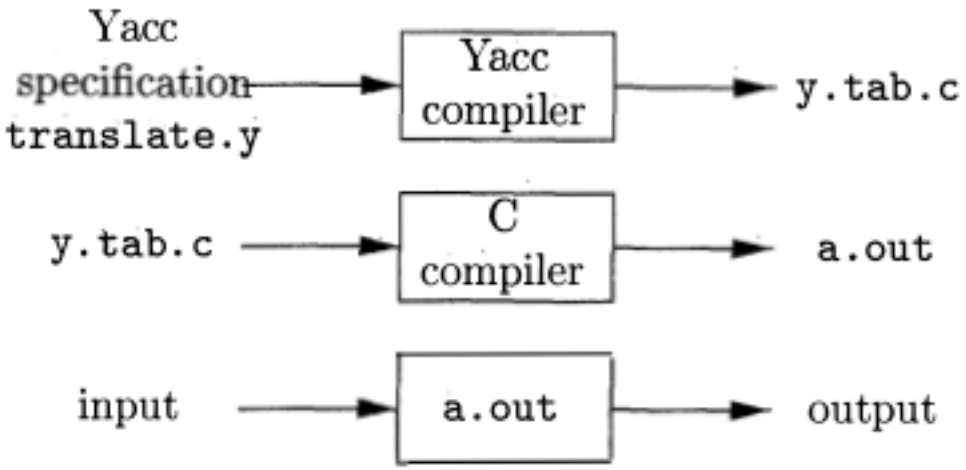
xianweiz.github.io

DCS292, 3/25/2021

中山大學 SUN YAT-SEN UNIVERSITY

Yacc Overview

- Yacc is an LALR(1) parser generator
  - YACC: Yet Another Compiler-Compiler
  - Parse a language described by a context-free grammar (**CFG**)
  - Yacc constructs an **LALR(1)** table
- Available as a command on the UNIX system
  - Bison: free GNU project alternative to Yacc



Yacc Specification

- **Definitions** section:
  - C declarations within %{ %}
  - Token declarations
- **Rules** section:
  - Each rule consists of a grammar production and the associated semantic action
- **Subroutines** section:
  - User-defined auxiliary functions

```
%{
  #include …
%}
%token NUM VAR
%%
production { semantic action }
…
%%
…
```

Write a Grammar in Yacc

- A set of productions <head> → <body>$_1$ | … | <body>$_n$ would be written in YACC as:

  <head> : <body>$_1$ { <semantic action>$_1$ }

  …

  : <body>$_n$ { <semantic action>$_n$ }

  ;

- Usages
  - Tokens that are single characters can be used directly within productions, e.g. ‘+’
  - Named tokens must be declared first in the declaration part using %token *TokenName*

Write a Grammar in Yacc (cont.)

- Semantic actions may refer to values of the synthesized attributes of terminals and non-terminals in a production:

  $X : Y_1\ Y_2\ Y_3\ \ldots\ Y_n$ { action }

  - $$ refers to the value of the attribute of X (non-terminal)
  - $i refers to the value of the attribute of Y_i (terminal or non-terminal)

- Example: $E \rightarrow E + T \mid T$

```
expr : expr '+' + term { $$ = $1 + $2 }
     | term
     ;
```

Default action: {$$ = $1 }

Example: E → E+E|E-E|E*E|E/E|(E)|num

```
1  %{
2   #include <ctype.h>
3   #include <stdio.h>
4   #define YYSTYPE double /* double type for Yacc stack */
5  %}
6  %token NUMBER
7
8  %left '+' '-'
9  %left '*' '/'
10
11 %%
12
13 lines : lines expr '\n' { printf("= %g\n", $2); }
14       | lines '\n'
15       | /* empty */
16       ;
17 expr : expr '+' expr { $$ = $1 + $3; }
18      | expr '-' expr { $$ = $1 - $3; }
19      | expr '*' expr { $$ = $1 * $3; }
20      | expr '/' expr { $$ = $1 / $3; }
21      | '(' expr ')' { $$ = $2; }
22      | NUMBER
23      ;
```

Can we remove those two lines?

Allow to evaluate a sequence of expressions, one to a line

Example (cont.)

```
24
25 %%
26
27 int yylex() {
28     int c;
29     while ((c = getchar()) == ' ') ;
30     if ((c == '.') || isdigit(c)) {
31         ungetc(c, stdin);
32         scanf("%lf", &yylval);
33         return NUMBER;
34     }
35     return c;
36 }
37
38 int main() {
39     if (yyparse() != 0)
40         fprintf(stderr, "Abnormal exit\n");
41     return 0;
42 }
43
44 int yyerror(char *s) {
45     fprintf(stderr, "Error: %s\n", s);
46 }
```

calls yylex() to get successive tokens

Compile and Run …

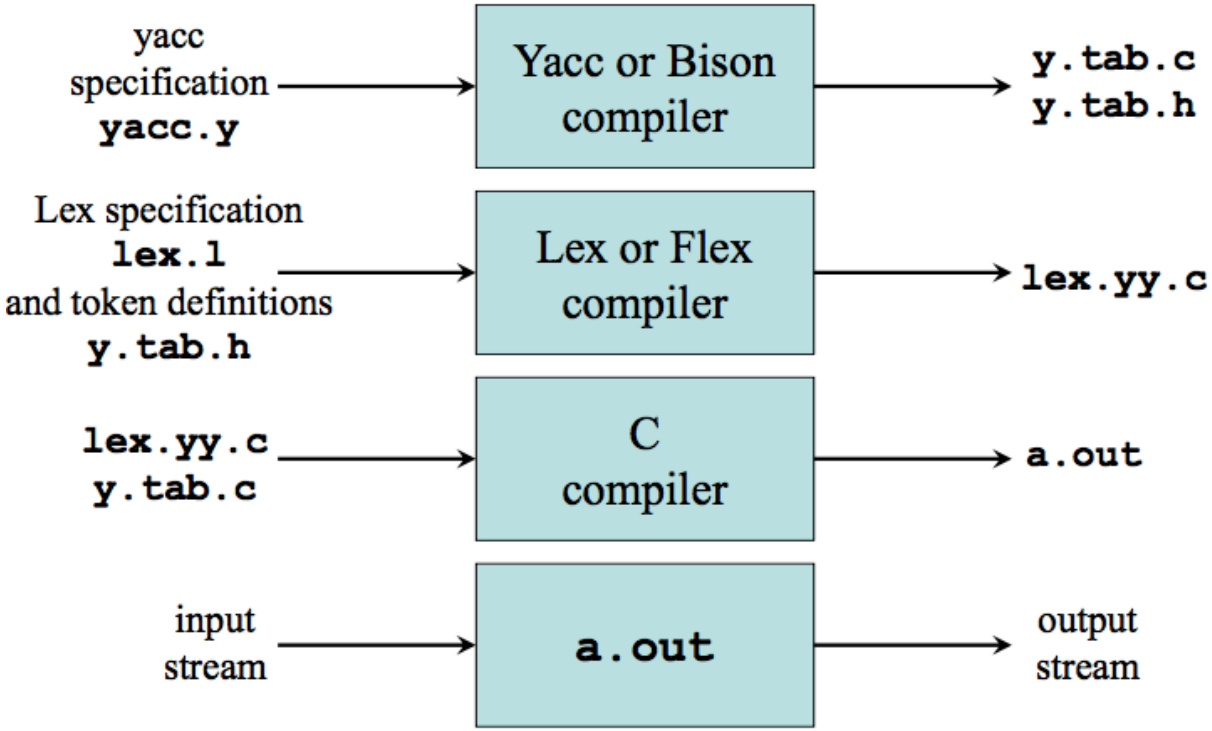
- Compile
  - $yacc -d *parser*.y
  - $gcc -o *test* y.tab.c

- Run
  - $./*test* < exprs.txt

```
1 1 + 5
2 1 * 2 + 10
3 10 - 2 -3
```

Yacc + Lex

- Lex was designed to produce lexical analyzers that could be used with Yacc
- Yacc generates a parser in y.tab.c and a header y.tab.h
- Lex includes the header and utilizes token definitions
- Yacc calls yylex() to obtain tokens



Example: Yacc + Lex

parser.y

```
%{
 #include <ctype.h>
 #include <stdio.h>
 #define YYSTYPE double /* double type for Yacc stack */
%}
%token NUMBER

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

%%

lines : lines expr '\n' { printf("= %g\n", $2); }
      | lines '\n'
      | /* empty */
      ;
expr : expr '+' expr { $$ = $1 + $3; }
     | expr '-' expr { $$ = $1 - $3; }
     | expr '*' expr { $$ = $1 * $3; }
     | expr '/' expr { $$ = $1 / $3; }
     | '(' expr ')' { $$ = $2; }
     | NUMBER
     ;

%%

/*
int yylex() {
    int c;
    while ((c = getchar()) == ' ') ;
    if ((c == '.') || isdigit(c)) {
        ungetc(c, stdin);
        scanf("%lf", &yylval);
        return NUMBER;
    }
    return c;
}
*/

int main() {
    if (yyparse() != 0)
        fprintf(stderr, "Abnormal exit\n");
    return 0;
}

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

lexer.l

```
%{
 #define YYSTYPE double
 #include "y.tab.h"
 extern double yylval;
%}
number [0-9]+\.?|[0-9]*\.[0-9]+

%%

[ ]         { /* skip blanks */ }
{number}    { sscanf(yytext, "%lf", &yylval);
              return NUMBER; }
\n|.        { return yytext[0]; }

%%

int yywrap(void) {
  return 1;
}
```

Generated by Yacc

Defined in y.tab.c

Compile and Run …

- Compile
  - $yacc -d *parser*.y
  - $lex *lexer*.l
  - $gcc -o *test* y.tab.c lex.yy.c

- Run
  - $./*test* < exprs.txt

```
1 1 + 5
2 1 * 2 + 10
3 10 - 2 -3
```

References

- 编译原理（第2版），章节4.9
- Yacc/Bison - Parser Generators, https://tldp.org/LDP/LG/issue87/ramankutty.html
- Lex and Yacc – A Quick Tour, https://courses.cs.washington.edu/courses/cse322/07au/slides/lec25.pdf
- ANTLR, Yacc, and Bison, https://www.cs.csustan.edu/~xliang/Courses/CS4300-20F/Notes/Ch4c.pdf
- Yacc Practice, https://epaperpress.com/lexandyacc/pry1.html
- The Lex & Yacc Page, http://dinosaur.compilertools.net/