

9. YACC

Yet Another Compiler-Compiler

충북대학교

이재성





학습내용

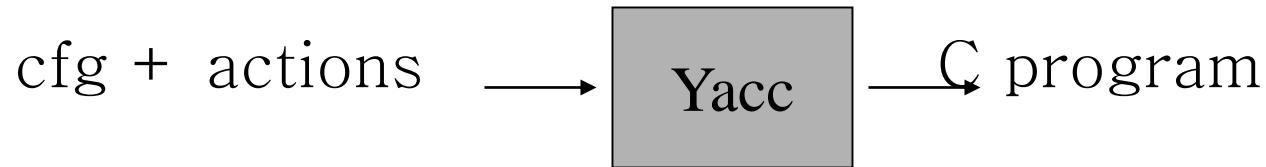
- Yacc 프로그램 개요
- Yacc 소스 프로그램 작성 방법
- Yacc 소스 예제(계산기)



개요

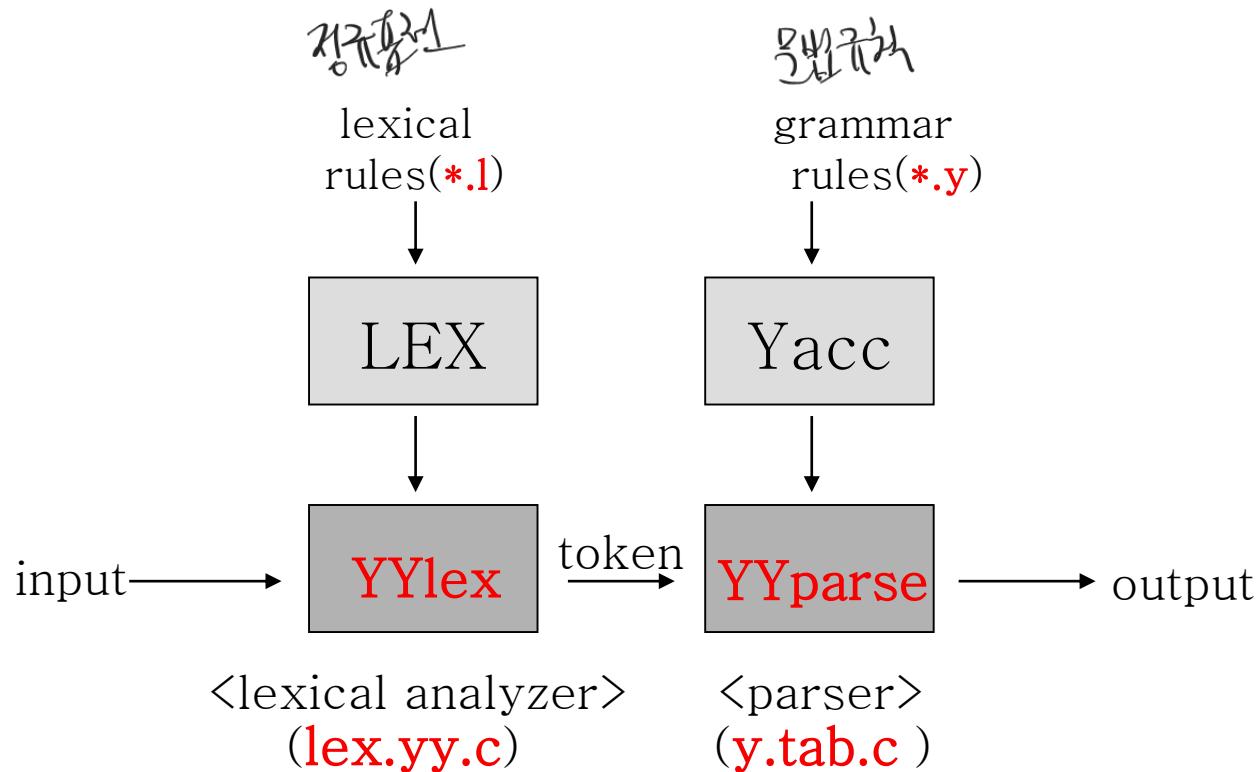
■ Yacc

- CFG와 각 문법에 해당하는 명령을 입력 받아 프로그램 생성





Lex와 Yacc의 모델





Review:

■ 파서 액션

- LR Parser : shift, reduce, accept, error.

- shift

- 파서는 어휘 분석기를 호출하여 토큰을 가져오고 이를 stack에 푸시함. 외부 변수 yylval은 value stack으로 복사

- reduce

- 그 rule에서 제공한 사용자 코드가 실행된다. 사용자 코드에서 return한 후, reduction이 수행(stack에 있는 handle이 축소되어 LHS의 비단말기호로 변경)

- stack이 현 상태를 유지하면서, value stack은 어휘 분석기와 규칙들과 관련된 동작들로부터 값을 가지며 평행하게 작동한다.



입력 명세

■ 형식:

declarations // 선언 부분

%%

rules // 생성 규칙 부분

%%

programs // 사용자 프로그램 부분



■ 선언 부분

- 생략될 수 있는 부분
- **%token** 토큰들을 나타내는 이름을 선언한다.
 - ex) %token name1 name2 ...
- **%start** 시작 심볼을 명시적으로 선언.
- default 시작 심볼은 규칙 부분의 첫 번째 생성 규칙의 LHS.

declarations
%%
rules
%%
programs

■ 생성 규칙 부분

- 형식: **A: RHS {액션 코드} ;**
- 문법규칙인 “**A:RHS**” 부분과 명령인 “**{액션 코드}**”로 구성

■ 사용자 프로그램 부분

- 생성된 프로그램 내부로 그대로 복사



생성 규칙 부분

■ 문법 규칙 + 액션 코드

- 각 문법 규칙에 사용자의 액션 코드를 연관
- 규칙은 입력 프로세스에서 인식된다.

■ 문법 규칙

- 형식

A : RHS

A: 하나의 비단말기호

RHS (Right Hand Side): 비단말기호, 토큰, 리터럴의 조합

ex) BNF $\langle \text{expression} \rangle ::= \langle \text{expression} \rangle + \langle \text{term} \rangle \mid \langle \text{term} \rangle$

YACC expression : expression '+' term | term



● 리터럴(literal):

- 따옴표가 붙은 문자열. 예: ‘+’
- C의 모든 escape sequences가 인식됨
ex) '**Wn**' newline '**Wb**' backspace '**Wt**' tab '**WW**' backslash

● 비단말기호의 이름 :

- 식별자와 같이 임의 길이로 된 문자열(처음 문자로 시작되고 그 뒤에 문자 또는 숫자들)
- 문자에는 점 “.”, underscore “_”를 포함
- 대문자와 소문자는 구별



-
- 수직 바 "|" 는 왼쪽을 다시 쓰는 것을 피하기 위해 사용할 수 있다.

ex) A : B C D ;

A : E F ; <----->

A : G ;

A : B C D

| E F

| G
;

- ϵ - 생성 규칙

ex) A $\rightarrow \epsilon$ <===> YACC A : ;



Annotated parse tree

액션 표현 APT (Semantic action) 만든 YAC 등기

- 액션은 중괄호 '{' 와 '}'에 둘러 쌓인 임의의 C 언어 표현
ex) expression : expression '+' term

```
{ printf("addition expression detected\n"); }  
;
```

expression : term

```
{ printf("simple expression detected\n"); }  
;
```

- 액션은 파스 트리(문법 트리)를 구성하거나 직접 코드를 생성하기 위해 사용 할 수 있다.
- YACC은 규칙의 중간과 끝에 액션을 허용한다.
- YACC 파서는 오직 **yy**로 시작하는 이름을 사용한다. 사용자는 이 형식의 이름을 피해야 한다.

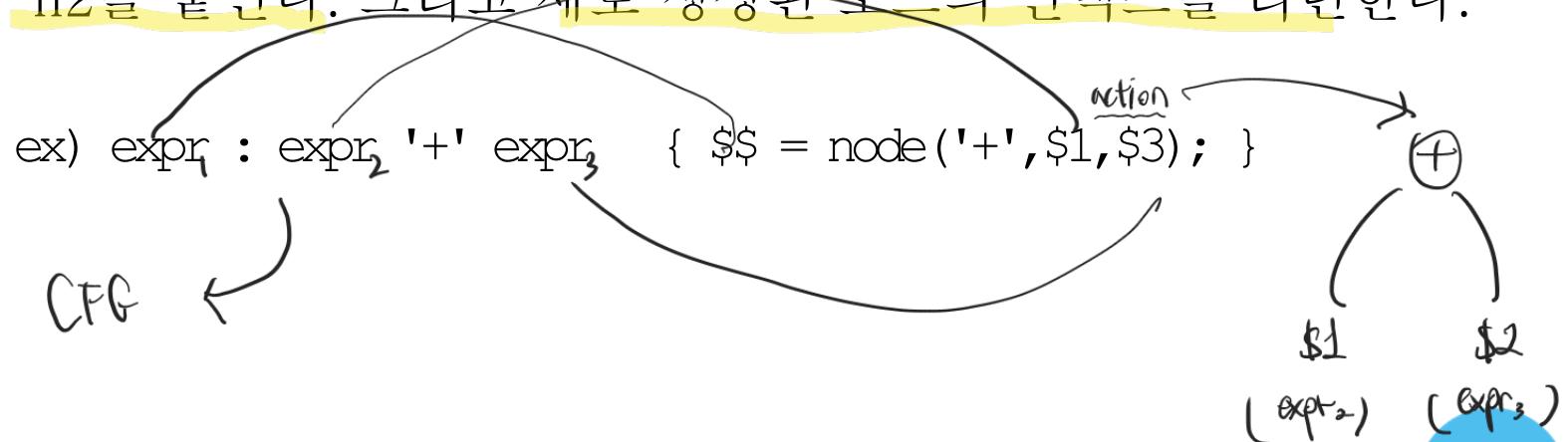


■ 문법 속성

- 문법 규칙에 있는 심볼들에 대한 값(속성)을 정의하기 위한 기능
- **\$\$, \$1, \$2, ...**: 각 문법 심볼의 속성 값을 나타낸다.
- $\$\$ = \text{LHS의 속성}$, $\$1 = \text{RHS 첫 심볼 속성}$, $\$2 = \text{RHS 둘째 심볼 속성}$

■ 파스 트리 구축 예

- **node(L,n1,n2)**은 레이블이 L인 노드를 생성하고, 자식 노드 n1과 n2를 붙인다. 그리고 새로 생성된 노드의 인덱스를 리턴한다.





모호성과 충돌

모호성

- 주어진 문법에 대해 두 개 이상의 파스트리가 구축될 수 있는 입력문자열이 있다면, 이 문법 규칙들은 모호하다 (ambiguous)고 한다.
- 파싱 도중 한 입력 토큰에 대해 shift와 reduce 액션이 2개 이상 가능한 경우 발생

Yacc 충돌 해결 규칙

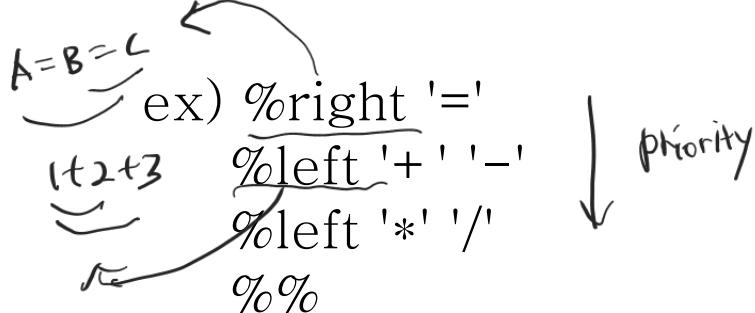
- shift/reduce 충돌에서, default는 shift를 실행
- reduce/reduce 충돌에서, default는 우선된(먼저 기술된) 문법 규칙에 따라 reduce를 실행

Shift는 무조건 제일 위에
서로 충돌이 날 수 있다.



우선순위

%left, %right, %nonassoc

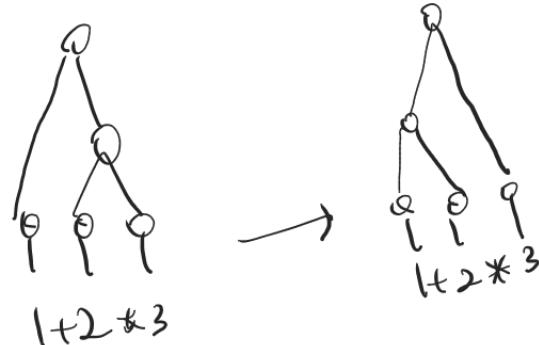


```
expr : expr '=' expr
      | expr '+' expr
      | expr '-' expr
      | expr '*' expr
      | expr '/' expr
;
```

$$a = b = ((c * d) - e) - (f * g)$$

\Leftrightarrow

$$a = (b = (((c * d) - e) - (f * g)))$$



- ① $c * d$
- ② $f * g$
- ③ $- e$
- ④ $- f$
- ⑤ $b =$
- ⑥ $a =$



Error 처리

■ 오류회복

- 오류가 발견 됐을 때 더 많은 구문 오류들을 찾기 위한 입력 스캐닝을 지속하는 것이 더 유용하다.

■ 토큰 error

- 오류 처리를 위해 예약된 것이며, 문법 규칙에 사용한다.
- 오류가 예상되는 위치에 사용되며, 토큰 ‘error’는 파서가 정상적인 상태에 들어갈 때까지 stack을 정리한다.



계산기 예제

■ 문제 : 정수 값에서 작동하는 기본적인 탁상용 계산기

■ calc.l

```
%{  
/* LEX source for calculator program */}  
%}  
%%  
[ \t] ; /* ignore blanks and tabs */ ) rule  
[-\r]* = [0-9]+ {yyval := atoi(yytext); return NUMBER;} D+  
"mod" return MOD; ASCII → int  
"div" return DIV;  
"sqr" return SQR;  
Wn|. return yytext[0]; /* return everything else */
```

수식
0 - 255
0 - 127

+ - * ? 아래나 등장.



■ calc.y 선언부

```
%{  
/* YACC source for calculator program */  
# include <stdio.h>  
%}  
%token NUMBER DIV MOD SQR  
%left '+' '-'  
%left '*' '/' DIV MOD  
%left SQR
```



calc.y 규칙

%%

```
comm : comm 'Wn'  
        | lambda  
        | comm expr 'Wn' {printf("%dWn", $2);} ;  
        | comm error 'Wn' {yyerrok; printf(" Try again Wn");}  
        ;  
  
expr : '(' expr ')' {$$ = $2;} ;  
      | expr '+' expr {$$ = $1 + $3;} ;  
      | expr '-' expr {$$ = $1 - $3;} ;  
      | expr '*' expr {$$ = $1 * $3;} ;  
      | expr '/' expr {$$ = $1 / $3;} ;  
      | expr MOD expr {$$ = $1 % $3;} ;  
      | SQR expr {$$ = $2 * $2;} ;  
      | NUMBER ;  
      ;  
  
lambda: /* empty */ ;
```

파서에서 오류가 발견되면 파서는
개행 문자로 지나간다. 오류상황을
reset(yyerrok)하고 적절한 메시지를
출력한다.



■ calc.y 사용자 루틴

```
%%  
#include "calclex.c"  
yyerror(s)  
char *s;  
{  
    printf("%s\n",s);  
}  
main()  
{  
    return yyparse();  
}
```



수행 예

컴파일 수행 예

```
% win_flex -wincompat -ocalclex.c -i calc.l
```

```
% win_bison -ocalc.c -d calc.y
```

```
% cc calc.c
```

수행 예

```
% calc
```

```
1+ 1
```

```
2
```

```
3+ 4*5
```

```
23
```

```
(3+ 4)*5
```

```
35
```

```
sqr sqr 2+ 3
```

```
19
```

```
(3))
```

```
syntax error
```

```
Try again
```

```
25 mod 7
```

```
4
```

```
^C
```

```
%
```



Flex Bison download

■ Download 위치

- <https://sourceforge.net/projects/winflexbison/>

■ 주요 스위치

- win_flex

--wincompat: 윈도우용 컴파일 호환 스위치

-o : 출력파일 이름 지정

- win_bison

-d : *.tab.h 파일 생성 -> lex 소스와 호환



참고 문헌

- Stephen C. Johnson, Yet Another Compiler-Compiler, Bell laboratories, Murry Hill, N.J. 07974, July 31, 1978
- Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, “Compilers – Principles, Techniques, and Tools,” Bell Telephone Laboratories, Incorporated, 1986.
- 오세만, “컴파일러 입문”, 정의사, 2004.