Lab02.Syntax Analysis

박성환

Syntax Analysis

- 실습 목표
  - LALR Parser 구현
  - 해당 파서로 소스코드를 읽어와 Syntax analysis 수행

Bison

- 파서 생성기(Parser Generator)s
- flex와 같은 구조를 가짐

```
%{
        선언부
%}


%%
        규칙부
%%
        사용자 프로그램
```

기본 예제

- 다음 문법을 인식하는 인식기 작성

S → CC
C → cC
C → d

기본 예제

basic.l

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

%%
"c" { return c; }
"d" { return d; }
\n { return EOL; }
[\t] { }
. { printf("Can not accept %c\n",
*yytext); }
%%
```

```
$> bison -d basic.y --debug
$> flex basic.l
$> gcc basic.tab.c lex.yy.c -lfl
```

basic.y

```
%{
#include <stdio.h>
%}
%token c d
%token EOL
%%
S: C C EOL { printf ("S -> C C
EOL\nAccepted\n"); }
;
C: c C { printf ("C -> c C\n"); }
 | d   { printf ("C -> d\n"); }
;
%%
```

```
int main (int argc, char *argv[])
{
 #ifdef YYDEBUG
    yydebug = 1;
 #endif
   yyparse();
}
yyerror (char *s)
{
   fprintf(stderr, "erros : %s\n", s);
}
```

기본 예제 (Mac)

basic.l

```
%{
#include "basic.tab.h“
%}

%%
"c" { return c; }
"d" { return d; }
\n { return EOL; }
[\t] { }
. { printf("Can not accept %c\n", *yytext); }
%%
```

basic.y

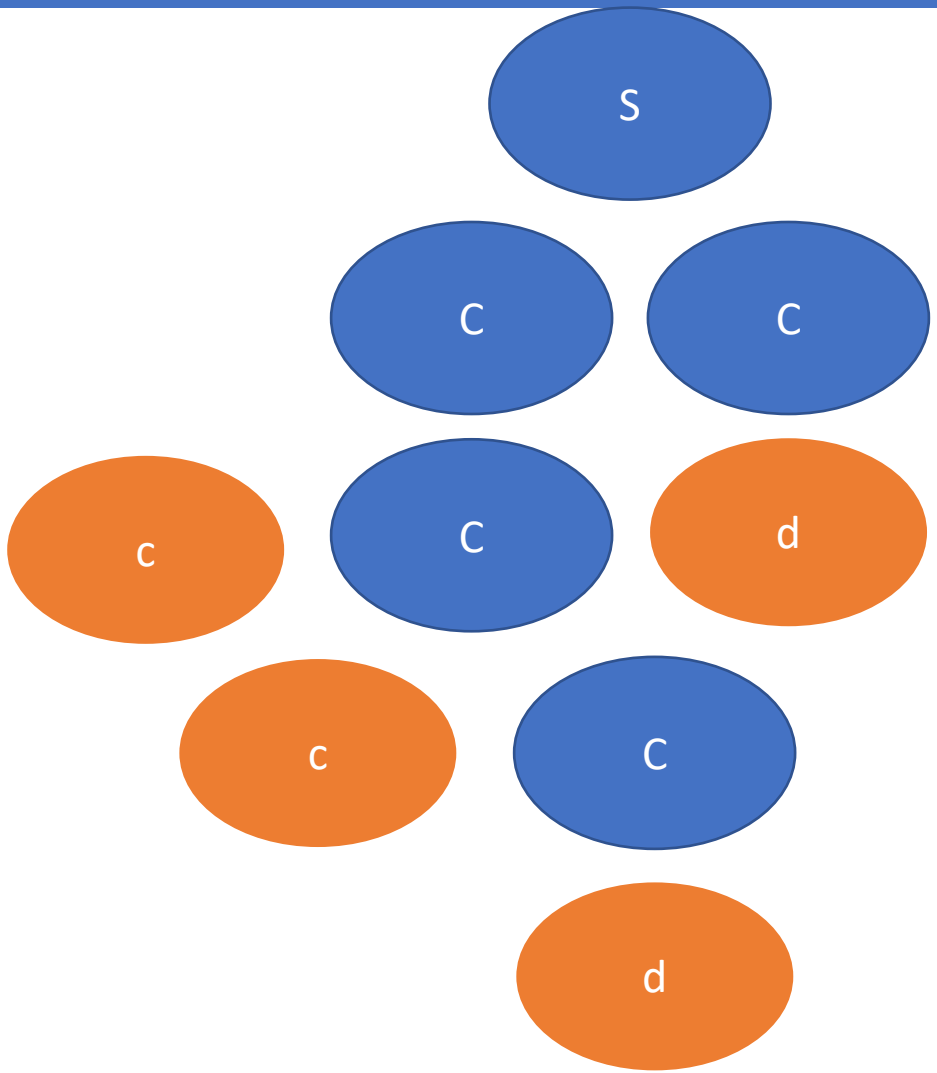
```
%{
#include <stdio.h>
int yylex();
void yyerror();
%}
%token c d
%token EOL
%%
S: C C EOL { printf ("S -> C C EOL\nAccepted\n"); }
;
C: c C { printf ("C -> c C\n"); }
 | d   { printf ("C -> d\n"); }
;
%%
```

```
int main (int argc, char *argv[])
{
 #ifdef YYDEBUG
    yydebug = 1;
 #endif
   yyparse();
}
void yyerror (char *s)
{
   fprintf(stderr, "erros : %s\n", s);
}
```

```
$> bison -d basic.y --debug
$> flex basic.l
$> gcc basic.tab.c lex.yy.c -lfl
```

Parse tree

- ccdd



계산기 만들기

comp.l

```
%{
#include "comp.tab.h“
%}

%%
"+" { return ADD; }
"-" { return SUB; }
"*" { return MUL; }
"/" { return DIV; }
"|" { return ABS; }
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
\n { return EOL; }
[\t] { /*ignore white space }*/ }
. { printf("Mystery character %c\n", *yytext); }
%%
```

```
$> bison -d comp.y
$> flex comp.l
$> gcc comp.tab.c lex.yy.c -lfl
```

comp.y

```
%{
#include <stdio.h>
int yylex();
void yyerror();
%}
%token NUMBER
%token ADD SUB MUL DIV ABS
%token EOL
%%
calclist: // Empty
   | calclist exp EOL { printf("= %d\n", $2); }
;
exp: factor
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term
   | factor MUL term { $$ = $1 * $3; }
   | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER
   | ABS term {$$ = $2 >= 0? $2 : -$2; }
;
%%
```

```
int main (int argc, char *argv[]){
   yyparse();
}
void yyerror(char *s){
   fprintf(stderr, "erros : %s\n", s);
}
```

계산기 만들기(Mac)

comp.l

```
%{
#include "comp.tab.h“
%}

%%
"+" { return ADD; }
"-" { return SUB; }
"*" { return MUL; }
"/" { return DIV; }
"|" { return ABS; }
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
\n { return EOL; }
[\t] { /*ignore white space }*/ }
. { printf("Mystery character %c\n", *yytext); }
%%
```

```
$> bison -d comp.y
$> flex comp.l
$> gcc comp.tab.c lex.yy.c -lfl
```

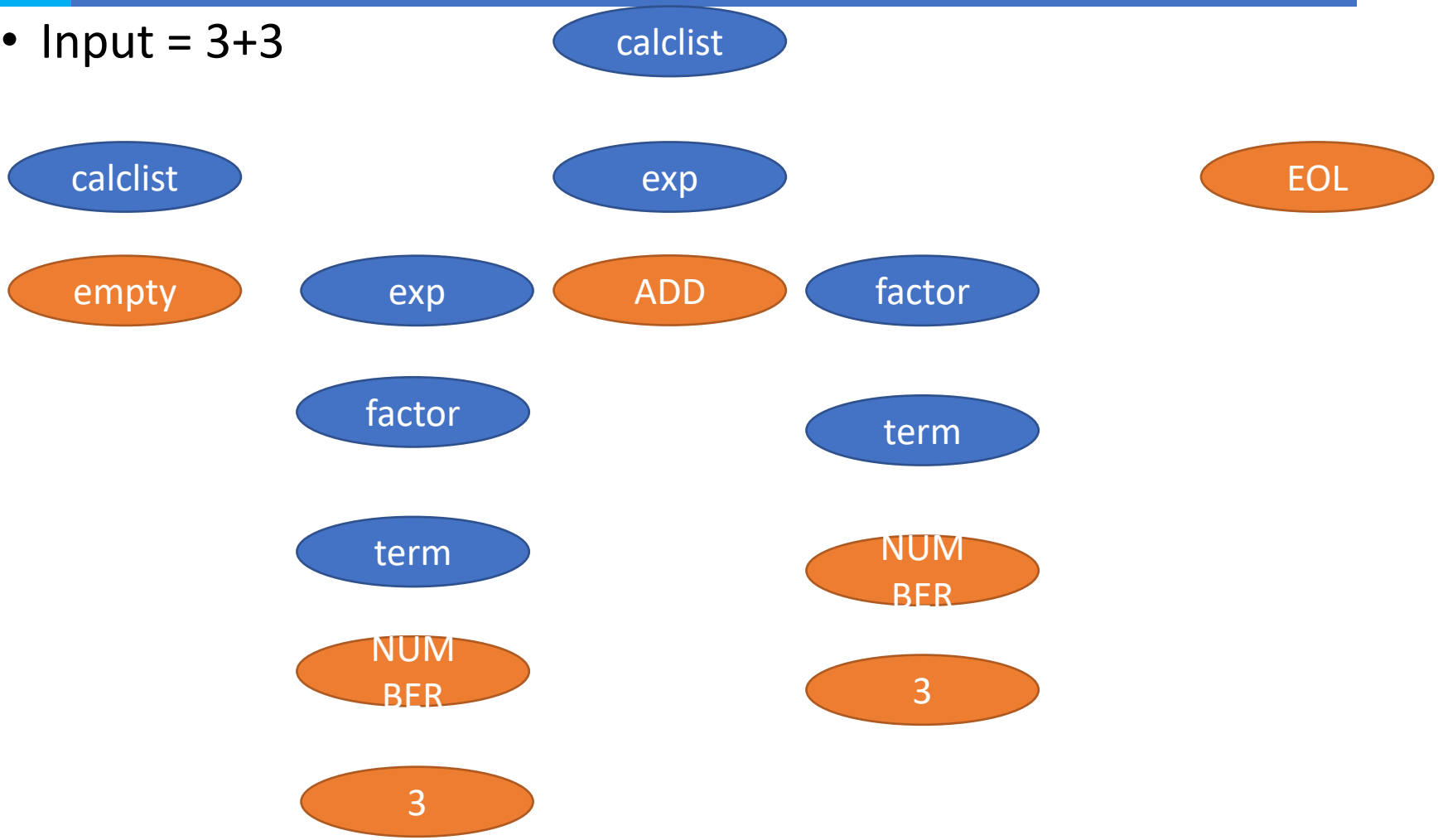
comp.y

```
%{
#include <stdio.h>
%}
%token NUMBER
%token ADD SUB MUL DIV ABS
%token EOL
%%
calclist: // Empty
   | calclist exp EOL { printf("= %d\n", $2); }
;
exp: factor
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term
   | factor MUL term { $$ = $1 * $3; }
   | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER
   | ABS term {$$ = $2 >= 0? $2 : -$2; }
;
%%
```

```
int main (int argc, char *argv[]){
   yyparse();
}
void yyerror(char *s){
   fprintf(stderr, "erros : %s\n", s);
}
```

Parse tree

- Input = 3+3



Grammar

- program -> block .
- block -> declList statement
- declList -> declList decl | ε
- decl -> constDecl | varDecl | funcDecl
- constDecl -> CONST numberList ;
- numberList -> IDENT EQ NUMBER
  | numberList COMMA IDENT EQ NUMBER
- varDecl -> VAR identList ;
- identList -> IDENT | identList COMMA IDENT
- optParList -> parList | ε
- parList -> IDENT | parList COMMA IDENT

Grammar

- funcDecl -> FUNCTION IDENT (optParList) block ;
- statement -> IDENT COLOEQ expression
  | BEGINN statement stateList END
  | IF condition THEN statement
  | WHILE condition DO statement
  | RETURN expression
  | WRITE expression
  | WRITELN
  | ε
- stateList -> stateList ; statement
  | ε

Grammar

- condition -> ODD expression
  | expression EQ expression
  | expression NOTEQ expression
  | expression LT expression
  | expression GT expression
  | expression LE expression
  | expression GE expression
- expression -> - term termList
  | term termList
- termList -> termList + term
  | termList - term
  | ε
- term -> factor factList
- facList -> factList * factor
  | factList / factor
  | ε

Grammar

- factor -> IDENT
  | NUMBER
  | IDENT (expList)
  | (expression)
- expList -> expression
  | expList , expression
  | ε
- COMMA -> ,

Syntax Analysis

hello.pl0

```
const n=5;
var x;
begin
    x:=n;
    write x;
    writeln
end.
```

result

```
jara@jara-System-Product-Name:~/ETC/CompilerLab/Lab02.SyntaxAnalysis/HW$ ./a.out < hello.pl0
numberList -> IDENT EQ NUMBER
constDecl -> CONST numberList ;
decl -> constDecl
declList -> declList decl
identList -> IDENT
varDecl -> VAR identList ;
decl -> varDecl
declList -> declList decl
factor -> IDENT
factList -> EMPTY
term -> factor factList
termList -> EMPTY
expression -> term termList
statement -> IDENT COLOEQ expression
factor -> IDENT
factList -> EMPTY
term -> factor factList
termList -> EMPTY
expression -> term termList
statement -> WRITE expression
stateList -> stateList ; statment
statemen -> WRITELN
stateList -> stateList ; statment
statement -> BEGIN statement stateList END
block -> declList statement
program -> block .
```

유의사항

- 10월 26일 23시 59분 까지 제출
- 주어진 lex file활용하여 과제 진행할 것
  - 따로 lex flie 작성하여 과제 진행할 시 해당 lex file 함께 제출 할 것
- yacc source code(.y)를 압축하여 다음과 같은 형식으로 제출
  - <학번>_Lab02.tar.gz
- 제출 기한 초과시 1일당 20% 감점

- tar 사용법
  - 압축
    - $ tar -zcvf <output_file> <input_file>
    - Ex) tar -zcvf 20202222_Lab02.tar.gz hw.y
  - 압축 해제
    - $ tar -zxvf <input_file>
    - Ex) tar -zxvf ex.tar.gz

Q&A