Project2: Yacc Programming

1. Grammar of subC

1.1 Tokens

다음은 Yacc과 Lex에 필요한 각 token들의 name이며 이는 subc.y 안의 declarations 에 작성 필요. 그 외 token들은 lex에서 yytext[0] 통해 token 문자를 그대로 리턴

Symbol Name	Symbol
TYPE	int, char
STRUCT	struct
SYM_NULL	NULL
RETURN	return
INTEGER_CONST	an integer constant
CHAR_CONST	a character constant
STRING	a string constant
IF	if
ELSE	else
WHILE	while
FOR	for
BREAK	break
CONTINUE	continue
LOGICAL_OR	||
LOGICAL_AND	&&
RELOP	<, <=, >, >=
EQUOP	==, !=
INCOP	++
DECOP	--
STRUCTOP	->

1.2 Operator Precedence and Associativity

다음은 각 token들의 precedence 및 associativity를 나타낸 것으로 이를 참고하여 subc.y 안의 declarations 에 각 토큰의 precedence 및 associativity 작성 필요 (**아래 표는 전체 token들에 관한 정보를 모두 포함하고 있지만 실제로 구현에서는 conflict와 관련된 token들의 precedence 및 associativity만 정의하면 충분함**)

Precedence	Operator	Name	Associativity
1	++ -- ( ) . ->	Suffix increment/decrement Function call and grouping Structure member access Structure member access through pointer	Left
2	++ -- - ! * &	Prefix increment/decrement Unary minus Logical NOT Indirection (dereference) Address-of	Right
3	* / %	Multiplication, division, and remainder	Left
4	+ -	Addition and subtraction	Left
5	< <= > >=	Less-than and less-than-or-equal Greater-than and greater-than-or-equal	Left
6	== !=	Equal and not-equal	Left
7	&&	Logical AND	Left
8	||	Logical OR	Left
9	=	Assignment	Right
10	,	Comma	Left

1.3 Productions

다음은 구현할 subc 의 Grammar에 관한 내용으로 이 Grammar를 수정없이 그대로 subc.y에 grammar rule 로 작성

```
program
  : ext_def_list
  ;

ext_def_list
  : ext_def_list ext_def
  | %empty
  ;

ext_def
  : type_specifier pointers ID ';'
  | type_specifier pointers ID '[' INTEGER_CONST ']' ';'
  | STRUCT ID '{' def_list '}' ';'
  | func_decl compound_stmt
  ;

type_specifier
  : TYPE
  | struct_specifier
  ;

struct_specifier
  : STRUCT ID '{' def_list '}'
  | STRUCT ID
  ;

func_decl
  : type_specifier pointers ID '(' ')'
  | type_specifier pointers ID '(' param_list ')'
  ;

pointers
  : '*'
  | %empty
  ;

param_list
  : param_decl
  | param_list ',' param_decl
  ;

param_decl
  : type_specifier pointers ID
  | type_specifier pointers ID '[' INTEGER_CONST ']'
  ;

def_list
  : def_list def
  | %empty
  ;

def
  : type_specifier pointers ID ';'
  | type_specifier pointers ID '[' INTEGER_CONST ']' ';'
  ;

compound_stmt
  : '{' def_list stmt_list '}'
  ;
```

```
stmt_list
  : stmt_list stmt
  | %empty
  ;

stmt
  : expr ';'
  | compound_stmt
  | RETURN expr ';'
  | ';'
  | IF '(' expr ')' stmt
  | IF '(' expr ')' stmt ELSE stmt
  | WHILE '(' expr ')' stmt
  | FOR '(' expr_e ';' expr_e ';' expr_e ')' stmt
  | BREAK ';'
  | CONTINUE ';'
  ;

expr_e
  : expr
  | %empty
  ;

expr
  : unary '=' expr
  | binary
  ;

binary
  : binary RELOP binary
  | binary EQUOP binary
  | binary '+' binary
  | binary '-' binary
  | binary '*' binary
  | binary '/' binary
  | binary '%' binary
  | unary %prec '='
  | binary LOGICAL_AND binary
  | binary LOGICAL_OR binary
  ;

unary
  : '(' expr ')'
  | INTEGER_CONST
  | CHAR_CONST
  | STRING
  | ID
  | '-' unary %prec '!'
  | '!' unary
  | unary INCOP %prec STRUCTOP
  | unary DECOP %prec STRUCTOP
  | INCOP unary %prec '!'
  | DECOP unary %prec '!'
  | '&' unary
  | '*' unary %prec '!'
  | unary '[' expr ']'
  | unary '.' ID
  | unary STRUCTOP ID
  | unary '(' args ')'
  | unary '(' ')'
  | SYM_NULL
  ;

args
  : expr
  | args ',' expr
  ;
```

2 Test

2.1 Output format

Project2 에서 만드는 Parser는 각 grammar rule이 인식 (reduce)되면 해당 rule에 대해 출력(printf)하는 방식으로 동작. test 폴더 안에 test c 코드 2개와 각각에 대한 출력 결과 result 파일이 있고, 실제로 만든 parser의 출력 결과가 비교

I/O Example

Input: test/test1.c

```
int main(){
 int a;
 char b;

 a = 10;
 b = 5;

 if ( a == 10 | | b == 5 ){
  return 1;
 } else {
  return 0;
 }
}
```

Output: test/test1_result

```
$ ./subc ../examples/open_test.c
ext_def_list ->epsilon
type_specifier ->TYPE
pointers ->epsilon
func_decl ->type_specifier pointers ID '(' ')'
def_list ->epsilon
type_specifier ->TYPE
pointers ->epsilon
def->type_specifier pointers ID ';'
def_list ->def_list def
type_specifier ->TYPE
pointers ->epsilon
def->type_specifier pointers ID ';'
def_list ->def_list def
stmt_list ->epsilon
unary ->ID
unary ->INTEGER_CONST
binary ->unary
expr ->binary
expr->unary '=' expr
stmt ->expr ';'
stmt_list ->stmt_list stmt
unary ->ID
unary ->INTEGER_CONST
binary ->unary
expr ->binary
expr->unary '=' expr
stmt ->expr ';'
stmt_list ->stmt_list stmt
unary ->ID
binary ->unary
unary ->INTEGER_CONST
```

```
binary ->unary
binary->binary EQUOP binary
unary ->ID
binary ->unary
unary ->INTEGER_CONST
binary ->unary
binary ->binary EQUOP binary
binary->binary LOGICAL_OR binary
expr->binary
def_list ->epsilon
stmt_list ->epsilon
unary ->INTEGER_CONST
binary ->unary
expr->binary
stmt->RETURN expr ';'
stmt_list ->stmt_list stmt
compound_stmt ->'{' def_list stmt_list '}'
stmt->compound_stmt
def_list ->epsilon
stmt_list ->epsilon
unary ->INTEGER_CONST
binary ->unary
expr->binary
stmt->RETURN expr ';'
stmt_list ->stmt_list stmt
compound_stmt ->'{' def_list stmt_list '}'
stmt->compound_stmt
stmt->IF '(' expr ')' stmt ELSE stmt
stmt_list ->stmt_list stmt
compound_stmt->'{' def_list stmt_list '}'
ext_def->func_decl compound_stmt
ext_def_list ->ext_def_list ext_def
program ->ext_def_list
```

3 제출

- lms 의 레포트 Project2 로 제출
- src 폴더 안에 작성한 파일들을 zip으로 압축하여 제출
- zip 파일 이름: project2_학번.zip (EX project2_202012345.zip)