1. 일반 선언문 (Declaration)

```
<declaration> ::= <declaration_specifiers>;
                | <declaration_specifiers> <init_declarator_list>;
<declaration_specifiers> ::= <type_specifier>
                           | <storage_class_specifier>
                           | <type_specifier> <declaration_specifiers>
                           | <storage_class_specifier> <declaration_specifiers>
<storage_class_specifier> ::= auto | static | typedef
<init_declarator_list> ::= <init_declarator>
                         | <init_declarator_list>, <init_declarator>
<init_declarator> ::= <declarator>
                    | <declarator> = <initializer>
<type_specifier> ::= <struct_specifier>
                   | <enum_specifier>
                   | <type_identifier>
<struct_specifier> ::= <struct_or_union> <identifier> {<struct_declaration_list>}
                     | <struct_or_union> {<struct_declaration_list>}
                     | <struct_or_union> IDENTIFIER
<struct_or_union> ::= struct | union
<struct_declaration_list> ::= <struct_declaration>
                            | <struct_declaration_list> <struct_declaration>
<struct_declaration> ::= <type_specifier> <struct_declarator_list>;
<struct_declarator_list> ::= <struct_declarator>
                           | <struct_declarator_list>, <struct_declarator>
<struct_declarator> ::= <declarator>
<enum_specifier> ::= enum IDENTIFIER { <enumerator_list> }
                   | enum { <enumerator_list> }
                   | enum IDENTIFIER
<enumerator_list> ::= <enumerator>
                    | <enumerator_list>, <enumerator>
<enumerator> ::= IDENTIFER
               | IDENTIFIER = <constant_expression>
<type_identifier> ::= INTEGER_TYPE_SPECIFER
                    | FLOATING_POINT_TYPE_SPECIFIER
                    | VOID_TYPE_SPECIFIER
                    | TYPEDEF_NAME
<declarator> ::= <pointer> <direct_declarator>
               | <direct_declarator>
<pointer> ::= * | * <pointer>
```

```
<direct_declarator> ::= IDENTIFIER
                        | ( <declarator> )
                        | <direct_declarator> [ <constant_expression_opt> ]
                        | <direct_declarator> ( <parameter_type_list_opt> )
<constant_expression_opt> ::= /* empty */
                              | <constant_expression>
<parameter_type_list_opt> ::= /* empty */
                              | <parameter_type_list>
<parameter_type_list> ::= <parameter_list>
                          | <parameter_list>, ...
<parameter_list> ::= <parameter_declaration>
                   | <parameter_list> , <parameter_declaration>
<parameter_declaration> ::= <declaration_specifiers> <declarator>
                            | <declaration_specifiers> <abstract_declarator>
                            | <declaration_specifiers>
<abstract_declarator> ::= <pointer>
                          | <direct_abstract_declarator>
                          | <pointer> <direct_abstract_declarator>
<direct_abstract_declarator> ::= ( <abstract_declarator> )
                                 | [ <constant_expression_opt> ]
                                 | ( <parameter_type_list_opt> )
                                 | <direct_abstract_declarator> [<constant_expression_opt>]
                                 | <direct_abstract_declarator> (<parameter_type_list_opt>)
<initializer> ::= <constant_expression>
                | { <initializer_list> }
<initializer_list> ::= <initializer>
                     | <initializer_list> , <initializer>
```

example 1) static int a[4] = { 1, 2, 3, 4 };
static : <storage_class_specifier>
int : <declaration_specifiers>
a[4] = {1, 2, 3, 4} : <init_declarator>
a[4] : <declarator>
{1, 2, 3, 4} : <initializer>

example 2) struct node { int a; int b; };
struct : <struct_or_union>
node : IDENTIFIER
int a; int b; : <struct_declaration_list>
int a; : <struct_declaration>
int b; : <struct_declaration>

example 3) enum day { mon, tue, wed = 2+1 };
day : IDENTIFIER;
mon, tue, wed = 2+1 : <enumerator_list>
mon : <enumerator>
tue : <enumerator>
wed = 2+1 : <enumerator>
2+1 : <constant_expression>

example 4) int *a;
int : <type_specifier>
* : <pointer>
a : <direct_declarator>
*a : <direct_declarator>

2. 수식 (Expression)

```
<expression> ::= <assignment_expression>
<assignment_expression> ::= <logical_or_expression>
                          | <unary_expression> = <assignment_expression>
<logical_or_expression> ::= <logical_and_expression>
                          | <logical_or_expression> || <logical_and_expression>
<logical_and_expression> ::= <equality_expression>
                          | <logical_and_expression> && <equality_expression>
<equality_expression> ::= <relational_expression>
                        | <equality_expression> == <relational_expression>
                        | <equality_expression> != <relational_expression>
<relational_expression> ::= <additive_expression>
                          | <relational_expression> < <additive_expression>
                          | <relational_expression> > <additive_expression>
                          | <relational_expression> <= <additive_expression>
                          | <relational_expression> >= <additive_expression>
<additive_expression> ::= <multiplicative_expression>
                          | <additive_expression> + <multiplicative_expression>
                          | <additive_expression> - <multiplicative_expression>
<multiplicative_expression> ::= <cast_expression>
                                | <multiplicative_expression> * <cast_expression>
                                | <multiplicative_expression> / <cast_expression>
                                | <multiplicative_expression> % <cast_expression>
<cast_expression> ::= <unary_expression>
                      | <type_name> <cast_expression>
<type_name> ::= <declaration_specifiers>
                | <declaration_specifiers> <abstract_declarator>
<unary_expression> ::= <postfix_expression>
                        | ++ <unary_expression>
                        | -- <unary_expression>
                        | & <cast_expression>
                        | * <cast_expression>
                        | ! <cast_expression>
                        | - <cast_expression>
                        | + <cast_expression>
                        | sizeof <unary_expression>
                        | sizeof <type_name>
<postfix_expression> ::= <primary_expression>
                         | <postfix_expression> [expression]
```

```
                        | <postfix_expression> ( <arg_expression_list_opt> )
                        | <postfix_expression>.IDENTIFIER
                        | <postfix_expression> -> IDENTIFIER
                        | <postfix_expression>++
                        | <postfix_expression>--
<arg_expression_list_opt> ::= /* empty */
                              | <arg_expression_list>
<arg_expression_list> ::= <assignment_expression>
                        | <arg_expression_list> , <assignment_expression>
<primary_expression> ::= IDENTIFIER
                        | INTEGER_CONSTANT
                        | FLOAT_CONSTANT
                        | CHARACTER_CONSTANT
                        | STRING_CONSTANT
                        | (<expression>)
```

example 1) a = b+10;
a = b+10 : <expression>
a : <unary_expression>
b+10 : <assignment_expression>

example 2) a = b.c || d->f
b || c : <logical_and_expression>
b.c : <postfix_expression>.IDENTIFIER
d->f : <postfix_expression> -> IDENTIFIER

3. C언어에 관해 잘못 알고 있었거나 모르고 있었던 것이 있으면 자유롭게 설명하시오.

평소 C언어 프로그램의 구조를 이해하고 그에 맞게 잘 사용하고 있다고 생각하였습니다. 하지만, 선언문, 명령문, 함수 등의 형태나 구조를 yacc 명세서 형식으로 설명하는 과정을 학습한 후에야 C언어의 문법과 의미를 정확하게 알고 있던 것이 아님을 깨달았습니다. 주교재의 질문 중 “while 문은 어떻게 구성되어 있는가?” 라고 질문하면 강의를 듣기 전엔 구조를 논리적으로 설명하지 못하고 예시를 들어 단순히 ‘이런 모양이다.’라고 대답하였습니다. 이렇게 명확한 구조를 파악하고 있지 않은 점이 평상시에 C언어에 관해 잘 모르고 있었던 것이었습니다.