declaration_specifiers

declaration_specifiers는 함수 선언문, 혹은 일반 선언문에서 declarator 앞에 오는 것으로 생략할 수 있습니다. 만약 이걸 생략하게 된다면 int로 간주됩니다. 예를 들어 f();라고 선언했으면 int f(); 라고 선언한 것과 같은 것입니다.

declaration_specifiers는 type_specifier, storage_class_specifier, type_ qualifier 가 하나 혹은 여러개가 연속해서 있는 것입니다.

type_specifier엔 struct_specifier, enum_specifier, TYPE_IDENTIFIER이 올 수 있습니다.
storage_class_specifier엔 auto, static, typedef, register, extern이 올 수 있습니다.

struct_specifier
-> struct_or_union(1) IDENTIFIER(2) { struct_declaration_list(3) }
| struct_or_union {struct_declaration_list}
| struct_or_union IDENTIFIER
ex) struct s {int a;} x;

(1) struct_or_union 은 struct | union 입니다.
(2) IDENTIFIER는 struct(union)의 이름이니다.
(3) struct_declaration_list 은 struct_declaration이 하나 혹은 여러개 오는 것입니다.
struct_declaration 은 type_specifier struct_declarator_list; 의 형식입니다.
- 일반 선언문과 다르게 storage_class_specifier가 올 수 없습니다. 전체가 일관되게 움직여야 하므로 어떤건 static, 어떤건 auto 이런 식일 수 없습니다.

struct_declarator_list는 struct_declarator가 하나 혹은 여러개 오는 것입니다.
- struct_declarator는 declarator와 같지만 초기화를 할 수 없습니다.

enum_specifier
-> enum IDENTIFIER {enumerator_list(1)} (선언)
| enum {enumerator_list} (선언)
| enum IDENTIFIER (참조)
ex) enum Color { red, green, blue };

struct_specifier와 유사한 모습이지만 struct_specifier는 struct를, enum_specifier는 enum을 선언할 때 사용합니다.
(1) enumerator_list는 enumerator가 하나 혹은 여러개 올 수 있습니다.
enumerator는 IDENTIFIER | IDENTIFIER = constant_expression의 형태입니다.
constant_expression은 *값*을 즉시 계산할 수 있는 수식을 의미합니다.

type_identifier
-> integer_type_identifier(1) | floating_point_type_specifier(2) | void_type_specifier(3) | typedef_name(4)
(1) integer_type_identifier는 signed, unsigned, char, int, long, short, char 등의 조합입니다.
(2) floating_point_type_identifier는 float, double, long double 입니다.
(3) void_type_specifier는 void 입니다.
(4) typedef_name은 프로그램 앞part에서 typedef로 미리 선언해둔 것입니다.
ex) typedef @@@ aa;
aa x; // aa는 typedef_name 에 해당

abstract_declarator

abstract_declarator는 declarator 중에서 이름이 빠진 것입니다. abstract_declarator는 함수를 선언할 때 쓰입니다.

함수를 선언할 때엔 parameter_type_list_opt가 필요한데
parameter_type_list_opt엔 parameter_list가 하나 이상으로 올 수 있고
parameter_list에는 parameter_declaration이 하나 혹은 여러개 올 수 있습니다.

parameter_declaration
-> declaration_specifiers declarator
| declaration_specifiers abstract_declarator
| declaration_specifiers 가 올 수 있습니다.

abstract_declarator
-> pointer
| direct_abstract_declarator
| pointer direct_abstract_declarator가 올 수 있습니다.

direct_abstract_declarator // direct_declarator에서 이름을 뺀 것입니다.
-> (abstract_declarator) // abstract_declarator가 괄호로 묶여 있거나,
| [constant_expression_opt] // 대괄호가 오거나 (상수 식은 생략 가능)
| (parameter_type_list_opt) // 소괄호가 오거나 (인자는 생략 가능)
| direct_abstract_declarator [constant_expression_opt]
| direct_abstract_declarator (parameter_type_list_opt) // 대괄호나 소괄호가 연속으로 오는 것입니다.

다시 정리해보자면
direct_declarator에서 이름을 뺀 형태인 [], (), 혹은 []과 ()가 연속해서 있는 것이 direct_abstract_declarator이고
direct_abstract_declarator 자체, 혹은 포인터가 붙어있는 것이 abstract_declarator입니다.
direct_declarator 자체거나 포인터가 붙어있는 것이 declarator이므로
declarator에서 이름을 뺀 것이 abstract_declarator이 되는 것입니다.

함수를 선언할 때 parameter_declaration의 예시에는
f(int arg1[10], int *arg2); ***…(1)***
f(int [10], int *); ***…(2)***
두 가지 방식이 있습니다.

(1)은 declaration_specifiers declarator 방식입니다. arg1 arg2이라는 **이름이 있기 때문**입니다.
(2)는 declaration_specifiers abstract_declarator 방식입니다. **이름이 없기 때문**이고 예시에서는 [10]과 *가 abstract_declarator인 것입니다.