**프로그래밍언어(가)**

**과제 4**

**이름 : 허예림**

**학번 : 20182667**

**교수님 : 유재우 교수님**

**1. 일반선언문(declaration)은 어떻게 생겼는지 예를 들어가며 설명하시오**

1-1

declaration

: declaration\_specifiers init\_declarator\_list ; ex) int x,y = 10;

Init – 초기화 기능

declaration\_specifiers -> 한 번 나와도 되고 여러 번 나와도 된다. 하지만, 문법적으로는 맞지만 진짜 사용하지는 못하는 경우도 있다. ex) float int x = 10;

: type\_specifier ex) int, float, long, …

| storage\_class\_specifier  
| type\_qualifier  
| type\_specifier declaration\_specifiers  
| storage\_class\_specifier declaration\_specifiers | type\_qualifier declaration\_specifiers

storage\_class\_specifier

: auto ex) (auto) int x;

| static ex) static int x;  
| typedef ex) typedef enum{false,true} BOOLEAN;

| register ex) register int x;

| extern

type\_qualifier

: const ex) const int x;

| volatile ex) volatile int x;

init\_declarator\_list

: init\_declarator

| init\_declarator\_list , init\_declarator ex) int x,y;

init\_declarator

: declarator  
| declarator = initializer ex) y = 10;

1-2

type\_specifier  
 : struct\_specifier ex) struct {…}

| enum\_specifier ex) enum {…}  
| TYPE\_IDENTIFIER ex) int x;

* integer\_type\_specifier:  
  signed, unsigned, char unsigned, short, long, int, char 등의 조합
* floating\_point\_type\_specifier float, double, long double
* void\_type\_specifier void
* typedef\_name ex) typedef int XX; XX y;  
  typedef 로 앞에 정의된 명칭

1-3

struct\_specifier  
: struct\_or\_union IDENTIFIER { struct\_declaration\_list } ex) struct s{…} -> 선언

| struct\_or\_union { struct\_declaration\_list } ex) struct { … } -> 선언  
| struct\_or\_union IDENTIFIER ex) struct a -> 참조

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 ex) {int x;} 가능, {const int x;} 불가능

struct\_declarator

: declarator  
| declaratoropt : expression

1-4

enum\_specifier  
: enum IDENTIFIER { enumerator\_list } ex) enum color{…}

| enum { enumerator\_list } ex) enum{…}  
| enum IDENTIFIER ex) enum color

enumerator\_list  
: enumerator

| enumerator\_list , enumerator

enumerator

: IDENTIFIER  
| IDENTIFIER = constant\_expression ex) enum{ A , B = 10;}

1-5

declarator  
: pointer direct\_declarator ex) int \*y;

| direct\_declarator

pointer

: \* type\_qualifier. ex) int \*const x ; const(int \*) x;  
| \* type\_qualifier pointer

+

int \*p - 정수형 타입을 가리키는 포인터  
int a[10] - 10개 짜리 정수의 배열

float \*f(); - 실수형 타입을 가리키는 포인터를 리턴하는 함수

float (\*fun) (); - 실수형 타입을 리턴하는 함수를 가리키는 포인터

1-6

direct\_declarator  
: IDENTIFIER ex) int \*y;

| ( declarator ) ex) int (\*y);  
| direct\_declarator [ constant\_expressionopt] ex) float \*b[10]

| direct\_declarator ( parameter\_type\_listopt ) ex) float \*b(int)

1-7

parameter\_type\_list  
: parameter\_list

| parameter\_list , ...

parameter\_list

: parameter\_declaration

| parameter\_list , parameter\_declaration ex) int b(A, B,…);

parameter\_declaration

: declaration\_specifiers declarator  
| declaration\_specifiers abstract\_declaratoropt

abstract\_declarator

: pointer

| pointeropt 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 )

1-8

initializer  
: constant\_expression ex) int x = 10;

| { initializer\_list }

initializer\_list

: initializer  
| initializer\_list , initializer

**2. 수식 (expression)은 어떻게 생겼는지 예를 들어가며 설명하시오**

primary\_expression

: IDENTIFIER

| INTEGER\_CONSTANT  
| FLOAT\_CONSTANT  
| CHARACTER\_CONSTANT

| STRING\_LITERAL

| ( expression )

postfix\_expression  
: primary\_expression

| postfix\_expression [ expression ] ex) kim[…]  
| postfix\_expression ( expression\_listopt ) ex) f()

| postfix\_expression . IDENTIFIER ex) kim.a  
| postfix\_expression -> IDENTIFIER ex) y->b  
| postfix\_expression ++ ex) a++  
| postfix\_expression – - ex) b--

arg\_expression\_list  
: assignment\_expression

| arg\_expression\_list , assignment\_expression

unary\_expression  
: postfix\_expression

| ++ unary\_expression ex) ++a

| -- unary\_expression. ex) --a  
| & cast\_expression ex) &a  
| \* cast\_expression ex) \*a

| ! cast\_expression. ex) !a  
| - cast\_expression. ex) -a  
| + cast\_expression. ex) a  
| sizeof unary\_expression

| sizeof ( type\_name )

cast\_expression  
: unary\_expression

| ( type\_name ) cast\_expression ex) (int) ~

type\_name

: declaration\_specifiers  
| declaration\_specifiers abstract\_declarator

Arithmetic Expressions

multiplicative\_expression

: cast\_expression

| multiplicative\_expression \* cast\_expression ex) T \* F

| multiplicative\_expression / cast\_expression ex) T / F

| multiplicative\_expression % cast\_expression ex) T % F

additive\_expression  
 : multiplicative\_expression

| additive\_expression + multiplicative\_expression

| additive\_expression - multiplicative\_expression

shift\_expression

: additive\_expression  
| shift\_expression << additive \_expression ex) a << b

| shift\_expression >> additive \_expression ex) a >> b

Relational Expressions

relational\_expression  
: shift\_expression

| relational\_expression < shift \_expression ex) a < b  
| relational\_expression > shift \_expression ex) a > b  
| relational\_expression <= shift \_expression ex) a <= b

| relational\_expression >= shift \_expression ex) a >= b

equality\_expression  
: relational\_expression

| equality\_expression == relational\_expression ex) a == b

| equality\_expression != relational\_expression ex) a != b

Logical Expression

logical\_and\_expression  
: equality\_expression

| logical\_and\_expression && equality\_expression ex) a && b

logical\_or\_expression

: logical\_and\_expression  
| logical\_or\_expression || logical\_and\_expression ex) a || b

conditional\_expression  
: logical\_or\_expression

| logical\_or\_expression ? expression : conditional\_expression ex) a ? b : c

Expression and Assignment

assignment\_expression  
: conditional\_expression

| unary\_expression = assignemnt\_expression ex) a =b , b = c

comma\_expression

: assignment\_expression

| comma\_expression , assignment\_expression ex) a,b

expression

: comma\_expression constant\_expression

: expression

**3. 여태까지 C의 문법과 관련하여 잘못 알고있었거나 모르고 있었던 사항이 있으면 모두 설명하시오**

교수님의 설명을 들으면서 가장 충격적이였던 부분은 If문을 설명할 때, 괄호 안에 있는 내용이 조건이 아닌 그저 수식이라는 점이였다. 추가적으로 statement의 case문에서도 조건이 아닌 수식으로 표현한다는 점을 알게 되었다. 지금까지는 무조건 조건문으로 작성해야한다고 알고 있었는데 교수님의 수업을 통해 그 부분이 수식으로 표현된다고 말해야 한다는 것을 알게 되었다. 그리고 declaration\_specifiers들을 실제로 사용할 수는 없지만 여러 번 사용해도 문법적으로 오류는 발생하지 않는다는 점을 알게 되었다. 또한, 가끔 수식 연산자들의 우선순위에 대해 헷갈리는 경우가 있었는데 이번 수업을 통해 큰 덩어리부터 공부하게 되니 자연스럽게 이해되는 부분들이 있어서 좋았다.