**컴파일러 과제4**

유재우교수님

소프트웨어학부 20163231 신승은

1. 일반선언문(declaration)은 어떻게 생겼는지 BNF규칙과 예를들어 설명

Declaration **//int x=0이나 int x,y,z=0과 같은 형태로 선언**

: declaration\_specifiers; **//int x=0**

| declaration\_specifiers init\_declarator\_list; **//int x,y,z=0;**

Declaration\_specifier

: type\_specifier

| storage\_class\_specifier

| type\_specifier declaration\_specifiers **//int 와 같은 형태**

| storage\_class\_specifier declaration\_specifiers **//static int 와 같은 형태**

Storage\_class\_specifier

: auto

| static

| typedef

Init\_declarator\_list

: init\_declarator **//int i**

| init\_declarator\_list, init\_declarator **//float x,y,z=10,4**

Init\_declarator

: declarator **//float f에서 f를 declarator라고 한다.**

| declarator = initializer **//float f=10.4에서 f=10.4를 말한다.**

Declarator

: pointer direct\_declarator **//int \*a에서 \*a를 말한다.**

| direct\_declarator **//int \*a에서 a를 말한다.**

Direct\_declarator

: IDENTIFIER

| ( declarator )

| direct\_declarator [ constant\_expression\_opt ]

| direct\_declarator ( parameter\_type\_list\_opt )

Constant\_expression\_opt

: /\* empty \*/

| constant\_expression **//int a[10]에서 a[10]을 말한다.**

Parameter\_type\_list\_opt

: /\* empty \*/

| parameter\_type\_list **//float ( \*b [10])(int)에서 (int)를 말한다.**

**// 실수형을 리턴하며 정수형 파라미터를 한 개 가진 함수를 가르키는 포인터의 배열**

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 **//주로 참조할 때 쓰는 표현**

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

**<struct\_specifier>의 예시**

Struct 신승은 {

Char \*name, address;

Int age, id;

} 승은1;

승은1 sxxgxxnz;

: struct\_specifier

<struct\_declaration>

<struct\_declarator\_list> : 각각 <struct\_declarator>를 나타낸다.

Enum\_specifier

: enum IDENTIFIER { enumerator\_list }

| enum { enumerator\_list }

| enum IDENTIFIER

Enumerator\_list

: enumerator

| enumerator\_list, enumerator

Enumerator

: IDENTIFIER

| IDENTIFIER = constant\_expres

**<enum\_specifier>의 예시**

Enum **fruit** { **apple, peach=10, grape=10+1, melon, mango** } ;

Enum fruit f1, f2;

**<enumerator\_list>의 하나하나 요소들을 <enumerator>라고 한다.**

**<expression>**

**<Identifier> : 써도되고 안써도됨**

<TYPE\_IDENTIFIER>

: intefer\_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 //typedef로 앞에 정의된 명칭

2. 수식(expression)은 어떻게 생겼는지 BNF규칙과 예를들어 설명

Primary\_expression

: IDENTIFIER

| INTEGER\_CONSTANT

| FLOAT\_CONSTANT

| CHARACTER\_CONSTANT

| STRING\_LITERAL

| (expression)

Postfix\_expression

: primary\_expression

| postfix\_expression [expression] **//char a[10+i]**

| postfix\_expression ( arg\_Expression\_list\_opt )

| postfix\_expression.IDENTIFIER **//승은1.age**

| postfix\_expression->IDENTIFIER **//승은1->address**

| postfix\_expression ++ **//후치수식 a++**

| postfix\_expression -- **//후치수식 a--**

Arg\_expression\_list\_opt

: /\* empty \*/

| arg\_expression\_list

Arg\_expression\_list

: assignment\_expression

| arg\_expression\_list, assignment\_expression

Unary\_expression

: postfix\_expression

| ++ unary\_expression **//전치수식**

| -- unary\_expression **//전치수식**

| & cast\_expression **// a & b에서 & b를 나타낸다.**

| \* cast\_expression **// a \* b에서 \* b 를 나타낸다.**

| ! cast\_expression **// switch(! B)에서 ! b를 나타낸다.**

| - cast\_expression **// a - b에서 - b를 나타낸다.**

| + cast\_expression **// a + b에서 + b를 나타낸다.**

| **sizeof** unary\_expression **//sizeof(num)의 형태로 num의 바이트를 계산한다.**

| **sizeof** (type\_name) **//sizeof(int)의 형태로 바이트 수를 알 수 있다.**

cast\_expression

: unary\_expression **//&a의 형태를 말한다.**

| (type\_name) cast\_expression **//(int) a의 형태를 말한다.**

Type\_name

: declaration\_specifiers **//int \*[10]에서 int를 말한다.**

| declaration\_specifiers abstract\_declarator **//int \*[10]에서 \*[10]를 abstract\_declarator라고 한다.**

**산술 연산**

Multiplicative\_expression

: cast\_expression

| multiplicative\_expression **\*** cast\_expression

| multiplicative\_expression **/** cast\_expression

| multiplicative\_expression **%** cast\_expression

Additive\_expression

: multiplicative\_expression

| additive\_expression **+** multiplicative\_expression

| additive\_expression **–** multiplicative\_expression

**관계 연산**

Relational\_expression

: additive\_expression

| relational\_expression **<** additive\_expression

| relational\_expression **>** additive\_expression

| relational\_expression **<=** additive\_expression

| relational\_expression **>=** additive\_expression

Equality\_expression

: relational\_expression

| equality\_expression **==** relational\_expression

| equality\_expression **!=** relational\_expression

**논리 연산**

Logical\_and\_expression

: equality\_expression

| logical\_and\_expression **&&** equality\_expression

Logical\_or\_expression

: logical\_and\_expression

| logical\_or\_expression **||** logical\_and\_exoression

Assignment\_expression

: logical\_or\_expression

| unary\_expression = assignment\_expression

Constant\_expression

: expression

Expression

: assignment\_expression

**<assignment\_expression>의 예시**

A = b = c+10

여기서 A, b는 unary\_expression이고, 은 assignment\_expression이자 expression이다.

3. 2장을 다 이해하고 난 후에 본인이 평상시에 C언어에 관해 잘못 알고 있었거나 모르고 있었던 것이 있으면 자유롭게 설명

19페이지에서 switch를 이용할 때 중괄호와 case를 꼭 썼었는데 그렇지 않아도 된다는 점을 이번에 알게 되었습니다.

20페이지에서 if 두 개와 마지막 else에서 모호성이 발생한다는 것을 이론적으로 봤을 때는 이해가 갔지만 평상시에 코드를 작성할 때는 전혀 인식하지 못했었습니다. 자동적으로 if와 else의 짝을 찾아 실행한다는 부분도 이번에 알게 되었습니다.