

# 프로그래밍언어론 hw3

C011013 권찬

2024.05.16

## 1 Yacc

### 1.1 YACC 개념

YACC (Yet Another Compiler Compiler) 는 유닉스 시스템의 표준 파서 생성기입니다.

YACC 는 보통 LEX 와 함께 쓰이는데, LEX 가 정규표현식을 기반으로 토큰을 생성하면, YACC 는 LEX 가 생성한 토큰을 기반으로 BNF 를 이용해 문법을 정의합니다.

YACC 는 grammar rule 과 각 룰에 따른 action 이 주어지면 이를 컴파일하여 C 코드로 구성된 Parser를 생성해줍니다.

LEX는 정규표현식을 기반으로 토큰을 생성하는 lex.yy.c 파일을 만듭니다. 이를 기반으로 yylex() 함수를 실행하여 어휘를 분석하고, 어휘를 분석하여 생성된 토큰은 yyparse() 함수의 입력으로 들어와 YACC에 정의된 문법 규칙대로 파싱됩니다.

### 1.2 YACC 동작 원리

YACC는 State Machine 으로 동작하여 주어지는 토큰들을 BNF 문법에 따라 차근차근 분석해나갑니다.

특정 상태에서 어떤 토큰이 들어오는지에 따라 각기 다른 상태로 천이되고, 이때 받을 수 있는 토큰들은 BNF 문법에 따라 정해져 있습니다.

만약 특정 상태에서 받을 수 있는 토큰이 아닌 토큰이 들어오면 Syntax Error를 발생시킵니다.

### 1.3 YACC 구현 설명

#### 1.3.1 Function

함수 정의, 함수 사용, printf 와 같은 내장 함수 사용 횟수를 카운팅합니다. 이 때 테스트 케이스에 함수가 선언만 되고 정의가 안된 경우는 없으며, 함수가 전방 선언되고 뒷 부분에 정의 된 경우, 뒷 부분에 대해서만 카운트 하는 점을 고려하면 함수의 전방 선언에 대해서는 변수를 세지 않고 무시하면 됩니다.

#### 1. 함수 정의 세기

함수의 정의는 yacc에서 function\_definition ⌈] external\_declaration 으로 reduce 될 때 세면 됩니다.

```
1     external_declaration
2       : function_definition {
3         ary[FUNC_CNT]++;
4
5         ary[INT_CNT] += intParameterCount;
6         intParameterCount = 0;
7
8         ary[CHAR_CNT] += charParameterCount;
9         charParameterCount = 0;
10
11        ary[POINTER_CNT] += pointerParameterCount;
12        pointerParameterCount = 0;
13    }
```

이때 함수 정의를 파악했다는 것은 함수 정의가 끝났다는 의미이므로, 지금까지 션 int, char, pointer 파라미터 개수도 최종 개수에 반영하도록 하였습니다. int, char, pointer 파라미터의 개수 카운팅은 뒤에서 정리하겠습니다.

## 2. 함수 호출 세기

```
1     postfix_expression
2       ...
3       | postfix_expression '(' ')'
4         ...
5           | postfix_expression '(' argument_expression_list ')'
6             ...
7               | ary[FUNC_CNT]++; /*함수 호출*/
8               | ary[FUNC_CNT]++; /*함수 호출*/ }
```

함수 호출은 postfix\_expression에서 reduce 되므로 이때 카운트 하였습니다.

## 3. 함수 포인터 세기 / 함수 전방 선언 무시

함수 포인터는 함수의 전방 선언과 그 느낌이 비슷하기 때문에 구분해주어야 했습니다.

```
1     direct_declarator
2       ...
3       | '(' declarator ')' {
4         if (isPointer) {
5           ary[POINTER_CNT]++;
6           isPointer = 0;
7           isPointerFunction = 1; // 포인터 함수 선언의 가능성이 있다.
```

```
8     } /* 무조건 변수를 나타낸다. */
9 }
```

우선 함수 포인터는 반드시 ”타입 (\*변수명)(인자 타입)”의 형태를 갖고 있기 때문에 ”(\*변수명)”이 나오면 함수 포인터 가능성이 있다고 보고 함수 포인터 체크를 활성화 하였습니다. 함수 포인터는 포인터와 함수를 모두 카운트해야 하는데, 우선 이 형식에서 포인터가 활성화 되었었다면, 함수 포인터가 아닌 경우 포인터 변수에 대한 선언이 되므로 최소한 포인터의 개수는 세는 것이 맞습니다.

```
1 direct_declarator
2 ...
3 | direct_declarator '(' parameter_list ')' {
4     if (isPointerFunction == 1) {
5         ary[FUNC_CNT]++;
6         isPointerFunction = 0;
7     } else if (isFunctionDeclaration == 0) {
8         isFunctionDeclaration = 1;
9     }
10 }
```

이때 만약 (파라미터) 형식의 데이터가 나와서 reduce 된다면 함수 포인터 변수를 선언한 것이므로, 함수 count도 증가시킵니다. 그렇지 않다면 함수에 대한 전방 선언으로 보고 전방 선언 체크를 활성화 합니다.

```
1 direct_declarator
2 ...
3 | direct_declarator '(' ')'
4   { if
5     → (isFunctionDeclaration == 0) isFunctionDeclaration = 1; }
```

함수의 전방 선언은 파라미터 없이도 될 수 있기 때문에 이 부분도 체크해줍니다.

```
1 external_declaration
2 ...
3 | declaration {
4     if (isFunctionDeclaration == 1) { // 함수 전방 선언은 무시한다.
5         #if YYDEBUG
6             printf("\n함수의 전방 선언은 무시합니다.\n");
7         #endif
8
9         intParameterCount = 0;
10        charParameterCount = 0;
11        pointerParameterCount = 0;
12    }
13    ...
```

최종적으로 declaration 이 external\_declaration 으로 reduce 될 때 함수의 전방 선언 체크 변수를 확인하여 만약 전방 선언이라면 파라미터의 개수를 세지 않도록 기존에 세어둔 값을 버립니다.

### 1.3.2 Operator

```
1     shift_expression
2       : additive_expression
3       | shift_expression LEFT_OP additive_expression { ary[OP_CNT]++; }
4       | shift_expression RIGHT_OP additive_expression { ary[OP_CNT]++; }
5       ;
6
7     additive_expression
8       : multiplicative_expression
9       | additive_expression '+' multiplicative_expression {
10          ↳ ary[OP_CNT]++; }
11       | additive_expression '-' multiplicative_expression {
12          ↳ ary[OP_CNT]++; }
13       ;
14
15     multiplicative_expression
16       : cast_expression
17       | multiplicative_expression '*' cast_expression { ary[OP_CNT]++; }
18       | multiplicative_expression '/' cast_expression { ary[OP_CNT]++; }
19       | multiplicative_expression '%' cast_expression { ary[OP_CNT]++; }
```

이항 연산자의 경우 이런식으로 카운팅하였습니다. 다른 이항 연산도 모두 동일하게 세었기 때문에 일부만 기술하였습니다.

```
1     unary_expression
2       : postfix_expression
3       | INC_OP unary_expression { ary[1]++; }
4       | DEC_OP unary_expression { ary[1]++; }
5       ...
6       ;
7
8     postfix_expression
9       ...
10    | postfix_expression '.' IDENTIFIER { ary[OP_CNT]++; }
11    | postfix_expression PTR_OP IDENTIFIER { ary[OP_CNT]++; }
12    | postfix_expression INC_OP { ary[OP_CNT]++; }
13    | postfix_expression DEC_OP { ary[OP_CNT]++; }
```

++, -- 연산자에 대해서도 과제 명세에 따라 카운팅하였습니다.

### 1.3.3 Int / Char / Pointer

먼저 현재 처리하고 있는 type 과 pointer 여부를 체크하기 위해 다음과 같이 체크 변수를 활성화 하였습니다.

```
1 typeSpecifier
2   : VOID
3   | CHAR {nowType = 1; /*char = 1*/}
4   | SHORT
5   | INT {nowType = 0; /* int = 0*/}
6   ...
```

nowType 변수를 이용하여 현재 type 을 확인하였습니다.

```
1 pointer
2   : '*' { isPointer = 1; }
3   | '*' typeQualifierList
4   | '*' pointer
5   | '*' typeQualifierList pointer
6   ;
```

pointer 여부는 위와 같이 \*를 만나면 곱셈 연산이 아닌 이상 반드시 포인터가 되므로 포인터 변수에 체크를 활성화하였습니다.

#### 1. 변수 세기

변수를 셀 때는

```
int a, b;
```

와 같이 여러 식별자가 나열될 수 있습니다. 이를 위해 현재 나열된 식별자의 개수를 저장하는 initVarCount 변수를 만들었습니다.

```
1 initDeclaratorList
2   : initDeclarator { initVarCount++; }
3   | initDeclaratorList ',' initDeclarator { initVarCount++; }
4   ;
```

initVarCount 변수에는 위와 같은 BNF 문법에서 식별자 리스트를 initDeclaratorList로 reduce 할 때 변수의 개수를 카운팅한 값을 저장하였습니다.

```
1 declaration
2   : declarationSpecifiers ';' 
3   | declarationSpecifiers initDeclaratorList ';' {
4     if (isTypeDef) {
5       ...
```

```

6 } else { // 아니라면 변수/함수에 대한 선언이다.
7     if (isPointer) {
8         ary[POINTER_CNT] += initVarCount;
9     }
10    if (nowType == 0) {
11        /* int 변수 카운트 */
12        #if YYDEBUG
13            printf("\nint 변수 선언: %d개\n\n", initVarCount);
14        #endif
15        ary[INT_CNT] += initVarCount;
16
17    } else if (nowType == 1) {
18        /* char 변수 카운트 */
19        #if YYDEBUG
20            printf("\nchar 변수 선언: %d개\n\n", initVarCount);
21        #endif
22        ary[CHAR_CNT] += initVarCount;
23    }
24
25    initVarCount = 0;
26    isPointer = 0;
27    nowType = -1;
28    isTypeDef = 0;
29
30 }
31 ;

```

식별자가 실제 변수로 선언되도록 reduce될 때 pointer 여부, int, char 여부를 확인하여 각각의 개수를 실제 카운팅 하였습니다.

## 2. 파라미터 세기

```

1 parameter_declarator
2 : declaration_specifiers declarator {
3     if (nowType == 0) {
4         intParameterCount++;
5     } else if (nowType == 1) {
6         charParameterCount++;
7     }
8
9     if (isPointer == 1) {
10        pointerParameterCount++;
11    }
12
13    nowType = -1;
14    isPointer = 0;
15 }

```

```

16 | declaration_specifiers abstract_declarator {
17 |   if (nowType == 0) {
18 |     intParameterCount++;
19 |   } else if (nowType == 1) {
20 |     charParameterCount++;
21 |   }
22 |
23 |   if (isPointer == 1) {
24 |     pointerParameterCount++;
25 |   }
26 |
27 |   nowType = -1;
28 |   isPointer = 0;
29 | }
30 | declaration_specifiers /* 파라미터 변수는 아니고 선언만 있으면 세지
31 | → 않는다. */ {
32 |   isPointer = 0;
33 |   nowType = -1;
34 | }
35 |
36 declaration_specifiers
37 ...
38 | typeSpecifier { isPointer = 0; } // 타입이 나오면 기존 포인터
39 | → 체크는 변수에 쓰인 포인터가 아님
40 ...

```

parameter\_declaration에서 파라미터 변수의 개수를 셹니다. 이때 int a(int, int)와 같이 선언하는 경우는 파라미터 '변수'를 선언한 것이 아니므로 과제 명세에 따라 세지 않도록 하였습니다. typeSpecifier 가 declarationSpecifier로 reduce 될 때도 pointer 체크를 비활성화 하였습니다. int a(int\*, int\*) 와 같은 경우를 처리하기 위함입니다.

### 3. 구조체에서 세기

```

1 struct_declarator_list
2   : struct_declarator {initVarCount++;}
3   | struct_declarator_list ',' struct_declarator {initVarCount++;}
4 ;

```

구조체에서도 마찬가지로 선언된 식별자들의 개수를 미리 세어둡니다.

```

1 struct_declaration
2   : specifier_qualifier_list struct_declarator_list ';' {
3     if (isPointer) {
4       ary[POINTER_CNT] += initVarCount;

```

```

5      }
6      if (nowType == 0) {
7          /* int 변수 카운트 */
8          #if YYDEBUG
9              printf("init int: %d\n\n", initVarCount);
10         #endif
11         ary[INT_CNT] += initVarCount;
12     } else if (nowType == 1) {
13         /* char 변수 카운트 */
14         #if YYDEBUG
15             printf("init char: %d\n\n", initVarCount);
16         #endif
17         ary[CHAR_CNT] += initVarCount;
18     }
19
20     initVarCount = 0;
21     isPointer = 0;
22     nowType = -1;
23 }
24 ;

```

그리고 구조체 내에서 변수가 선언될 때, 지금까지 세어둔 식별자들을 각각의 타입에 맞춰 카운팅합니다.

#### 1.3.4 Array

```

1 direct_declarator
2 ...
3 | direct_declarator '[' constant_expression ']' { isArray = 1; }
4 | direct_declarator '[' ']' { isArray = 1; }
5 ...

```

배열의 경우 대괄호가 있으면 배열과 관련된 연산이므로 배열 체크 변수를 활성화 합니다.

```

1 declarator
2 : pointer direct_declarator { if (isArray) {ary[ARRAY_CNT]++;
3     ↳ isArray = 0;} }
4 | direct_declarator           { if (isArray) {ary[ARRAY_CNT]++;
5     ↳ isArray = 0;} }
6 ;

```

배열 체크 변수가 활성화 되었을 때 변수가 선언되었다면 배열 변수로 카운팅 합니다.

### 1.3.5 Selection / 반복문

```
1   statement_list
2     : statement           { if (isSelection)
3       ↳ {ary[SELECTION_CNT]++; isSelection = 0; } }
4     | statement_list statement { if (isSelection)
5       ↳ {ary[SELECTION_CNT]++; isSelection = 0; } }
6     ; // OK
7
8   statement
9     ...
10    | selection_statement { isSelection = 1; }
11    | iteration_statement { ary[LOOP_CNT]++; }
12    | jump_statement
13    ;
```

Selection 을 셀 때는, if 로 구성된 구문과 switch로 구성된 구문만 세어야 했습니다.

따라서 selection\_statement에서 바로 세는 것이 아니라, selection\_statement 가 등장하면 selection 체크 변수를 설정하고, statement 가 끝이 났을 때 그 때 반복문 개수를 세도록 하였습니다.

반복문은 iteration\_statement 를 만났을 때 바로 세도록 하였습니다.

### 1.3.6 Return문

```
1   jump_statement
2     : GOTO IDENTIFIER ';'
3     | CONTINUE ';'
4     | BREAK ';'
5     | RETURN ';'          { ary[RETURN_CNT]++; }
6     | RETURN expression ';' { ary[RETURN_CNT]++; }
7     ; // OK
```

return문은 jump\_statement에서만 등장하기 때문에 이곳에서 카운팅 해주었습니다.

### 1.3.7 변수의 선언 위치 자유

```
1   compound_statement
2     : '{' '}'
3     | '{' statement_list '}'
4     /* | '{' declaration_list '}' // declaration_statement // test
```

```

5   | '{' declaration_list statement_list '}' // 여기에서 둘 사이에
6   |→ 순서를 없애주어야 함. */
; // OK

```

강의록에 있는 C 언어 BNF 문법에서는 { declaration\_list statement\_list } 부분 때문에 선언하는 부분이 항상 statement 보다 먼저 나와야 했습니다. 변수의 선언 위치를 자유롭게 해주기 위해서는 이 둘 사이에 존재하는 순서 관계를 없애줄 필요가 있었습니다.

```

1  statement
2  : labeled_statement
3  | compound_statement
4  | declaration_or_expression_statement
5  | selection_statement { isSelection = 1; }
6  | iteration_statement { ary[LOOP_CNT]++; }
7  | jump_statement
8  ;
9
10 declaration_or_expression_statement
11 : expression_statement
12 | declaration
13 ;

```

이를 위해 기존 expression\_statement 대신에 declaration\_or\_expression\_statement라는 부분을 위와 같이 새로 정의하였습니다.

```

1  compound_statement
2  : '{' '}'
3  | '{' statement_list '}'
4  ;

```

그리고 compound\_statement에서는 기존에 문제가 되었던 declaration 관련 부분을 모두 없애주었습니다.

```

1  iteration_statement
2  : WHILE '(' expression ')' statement
3  | DO statement WHILE '(' expression ')' ';' ;
4  ||| FOR '(' expression_statement expression_statement ')'
5  |→ statement
6  ||| FOR '(' expression_statement expression_statement expression
7  |→ ')' statement
8  | FOR '(' declaration_or_expression_statement expression_statement
9  |→ ')' statement

```

```
7 | FOR '(' declaration_or_expression_statement expression_statement
8 |→ expression ')' statement
9 ;
```

마지막으로 반복문에서도 for(int i = 0; i < 10; i++) 과 같은 문장을 처리할 수 있도록 제한을 풀어주었습니다.

### 1.3.8 #include

```
1 ...
2 INCLUDE #(include) .*\n
3 ...
4 %%
5 ...
6 {INCLUDE} {; }
7 ...
8 ...
```

#include는 별도 기능을 처리하지 않도록 lex에서 토큰을 생성하지 않고 무시해 주었습니다.

### 1.3.9 #define

```
1 ...
2 %%
3 ...
4 #define {return DEFINE;}
5 ...
```

#define은 먼저 lex에서 토큰을 따로 생성해주었습니다.

```
1 external_declaraction
2 ...
3 | DEFINE IDENTIFIER CONSTANT { // "#define 식별자 상수" 처리
4 |   #if YYDEBUG
5 |     printf("add define : %s\n\n", identifier_name);
6 |   #endif
7 |   define_name_array[define_name_array_size++] = identifier_name;
8 |
9 | | DEFINE IDENTIFIER STRING_LITERAL { // "#define 식별자 문자열" 처리
10 | |   #if YYDEBUG
11 | |     printf("add str define : %s\n\n", identifier_name);
12 | |   #endif
```

```

13     str_define_name_array[str_define_name_array_size++] =
14         ↳ identifier_name;
}

```

yacc에서는 external\_declaration에 #define을 인식하도록 BNF를 위와 같이 추가하였습니다. 이때 #define 구문을 인식하면, lex에서 저장한 식별자 이름을 #define 심볼 테이블에 상수, 문자열로 구분하여 저장합니다.

```

1 {L}({L}|{D})* {
2 ...
3 // 식별자가 현재 #define constant 심볼 테이블에 있는지 확인
4 for (int i = 0; i < define_name_array_size; i++) {
5     if (strcmp(identifier_name, define_name_array[i]) == 0) {
6         #if DEBUG
7             printf("define name : %s\n\n", identifier_name);
8         #endif
9         return CONSTANT;
10    }
11 }
12 ...
13 // 식별자가 현재 #define 문자열 리터럴 심볼 테이블에 있는지 확인
14 for (int i = 0; i < str_define_name_array_size; i++) {
15     if (strcmp(identifier_name, str_define_name_array[i]) == 0) {
16         #if DEBUG
17             printf("define name : %s\n\n", identifier_name);
18         #endif
19         return STRING_LITERAL;
20    }
21 }
22 ...
23 }

```

lex에서는 이후에 식별자를 읽을 때마다 기존에 저장했던 #define 심볼 테이블에 식별자가 들어있는지 파악해서 CONSTANT 또는 STRING\_LITERAL 토큰을 돌려줍니다.

### 1.3.10 기타 키워드

#### 1. **typedef**

typedef도 #define과 유사하게 처리합니다.

```

1 ...
2 declaration
3     : declaration_specifiers ';'

```

```

4 | declaration_specifiers init_declarator_list ';' {
5 |   if (isTypeDef) { // typedef 키워드가 등장했었다면, 새로운 타입에
6 |     ↪ 대한 선언이다.
7 |     #if YYDEBUG
8 |       printf("add type : %s\n\n", identifier_name);
9 |     #endif
10 |     type_name_array[type_name_array_size] = identifier_name;
11 |     type_name_array_size++;
12 |     isTypeDef = 0;
13 |   } else { // 아니라면 변수/함수에 대한 선언이다.
14 |     ...
15 |   }
16 |   isTypeDef = 0;
... 
```

typedef 도 #define 과 유사하게 처리합니다. 만약 typedef 선언이 등장했다면, lex에서 읽은 식별자 이름을 type name 심볼 테이블에 등록합니다.

```

1 {L}({L}|{D})* {
2   identifier_name = strdup(yytext);
3
4   // type name symbol table에 현재 읽은 식별자가 있는지 확인
5   for (int i = 0; i < type_name_array_size; i++) {
6     if (strcmp(identifier_name, type_name_array[i]) == 0) {
7       return TYPE_NAME;
8     }
9   }
10
11 ...
12 } 
```

lex에서 식별자를 읽었을 때, 만약 type name 심볼 테이블에 존재하는 식별자라면 TYPE\_NAME 토큰을 돌려주도록 하였습니다.

```

1 typeSpecifier
2   : VOID
3   | CHAR {nowType = 1; /*char = 1*/}
4   | SHORT
5   | INT {nowType = 0; /* int = 0*/}
6   | LONG
7   | FLOAT
8   | DOUBLE
9   | SIGNED
10  | UNSIGNED
11  | struct_or_union_specifier
12  | enum_specifier 
```

```
13     | TYPE_NAME  
14     ;  
15 }
```

그렇게 인식한 TYPE\_NAME 토큰은 다른 type 토큰들과 동일하게 처리됩니다.

### 1.3.11 주석문

```
1 ...  
2 COMMENT (\n|\r|/).*\n|(\n|\r|(.|\[\r\n])*)?\n*/  
3 %%  
4 {COMMENT} {}  
5 #define {return DEFINE;}  
6 "char" {return CHAR;}  
7 "short" {return SHORT;}  
8 ...
```

주석은 소스 코드 안에서 모든 코드를 처리하지 않도록 해야 하므로, lex에서 제일 먼저 토큰으로 잡아주었습니다.

## 2 소스코드

### 2.1 Lex

```
1 %{  
2 #include <stdio.h>  
3 #include <string.h>  
4 #include "y.tab.h"  
5 // #define DEBUG 1  
6  
7 // typedef 심볼테이블  
8 extern char* type_name_array[1000];  
9 extern int type_name_array_size;  
10  
11 // #define constant 심볼테이블  
12 extern char* define_name_array[1000];  
13 extern int define_name_array_size;  
14  
15 // #define string literal 심볼테이블  
16 extern char* str_define_name_array[1000];  
17 extern int str_define_name_array_size;  
18  
19 // 현재 보고 있는 식별자의 이름  
20 extern char* identifier_name;
```

```

21    %}
22
23    D [0-9]
24    L [a-zA-Z_]
25    H [a-fA-F0-9]
26    E [Ee] [+ -] ?{D}+
27    FS (f|F|l|L)
28    IS (u|U|l|L)*
29    INCLUDE #\include.*\n
30    COMMENT (\/\*.*\n|(\/\*\*(.|[\r\n])*?\*/\*)
31
32    /**
33    {COMMENT} {}
34    #define {return DEFINE;}
35    "char" {return CHAR;}
36    "short" {return SHORT;}
37    "int" {return INT;}
38    "long" {return LONG;}
39    "signed" {return SIGNED;}
40    "unsigned" {return UNSIGNED;}
41    "float" {return FLOAT;}
42    "double" {return DOUBLE;}
43    "const" {return CONST;}
44    "volatile" {return VOLATILE;}
45    "void" {return VOID;}
46
47    "auto" {return AUTO;}
48
49    "enum" {return ENUM;}
50    "register" {return REGISTER;}
51    "static" {return STATIC;}
52
53    "break" {return BREAK;}
54    "case" {return CASE;}
55    "continue" {return CONTINUE;}
56    "default" {return DEFAULT;}
57    "do" {return DO;}
58    "else" {return ELSE;}
59    "extern" {return EXTERN;}
60    "for" {return FOR;}
61    "goto" {return GOTO;}
62    "if" {return IF;}
63    "return" {return RETURN;}
64    "sizeof" {return SIZEOF;}
65
66    "struct" {return STRUCT;}
67    "switch" {return SWITCH;}
68    "typedef" {return TYPEDEF;}
69    "union" {return UNION;}
70    "while" {return WHILE;}

```

```

71 {L}({L}|{D})* {
72     // 식별자를 발견했을 때, 우선 토큰으로 잡은 이름을 identifier_name에
73     // 저장한다.
74     // strdup() 함수는 yytext가 가리키는 문자열을 다른 별도의 메모리
75     // 공간으로 복사한다
76     // yytext에는 매번 현재 읽은 토큰의 값이 들어가므로, 그 포인터 주소를
77     // 바로 저장할 수 없기 때문이다.
78     identifier_name = strdup(yytext);
79
80 #if DEBUG
81     printf("identifier: %s\n\n", identifier_name);
82 #endif
83
84     // type name symbol table에 현재 읽은 식별자가 있는지 확인
85     for (int i = 0; i < type_name_array_size; i++) {
86         if (strcmp(identifier_name, type_name_array[i]) == 0) {
87             #if DEBUG
88                 printf("type name : %s\n\n", identifier_name);
89             #endif
90             return TYPE_NAME;
91         }
92     }
93
94     // 식별자가 현재 #define constant 심볼 테이블에 있는지 확인
95     for (int i = 0; i < define_name_array_size; i++) {
96         if (strcmp(identifier_name, define_name_array[i]) == 0) {
97             #if DEBUG
98                 printf("define name : %s\n\n", identifier_name);
99             #endif
100            return CONSTANT;
101        }
102    }
103
104    // 식별자가 현재 #define 문자열 리터럴 심볼 테이블에 있는지 확인
105    for (int i = 0; i < str_define_name_array_size; i++) {
106        if (strcmp(identifier_name, str_define_name_array[i]) == 0) {
107            #if DEBUG
108                printf("define name : %s\n\n", identifier_name);
109            #endif
110            return STRING_LITERAL;
111        }
112    }
113
114    0[xX]{H}+{IS}? {return CONSTANT;}
115    0{D}+{IS}? {return CONSTANT;}
116    {D}+{IS}? {return CONSTANT;}
117    L?(\[\\].|[^\\])+? {return CONSTANT;}

```

```

118 {D}+{E}{FS}? {return CONSTANT;}
119 {D}*". "{D}+({E})?{FS}? {return CONSTANT;}
120 {D}+". "{D}*({E})?{FS}? {return CONSTANT;}
121
122 L?"\\.|[^\\"])*" {return STRING_LITERAL;}
123
124 "..." {return ELLIPSIS;}
125
126 ">==" {return RIGHT_ASSIGN;}
127 "<==" {return LEFT_ASSIGN;}
128 "+=" {return ADD_ASSIGN;}
129 "-=" {return SUB_ASSIGN;}
130 "*=" {return MUL_ASSIGN;}
131 "/=" {return DIV_ASSIGN;}
132 "%=" {return MOD_ASSIGN;}
133 "&=" {return AND_ASSIGN;}
134 "^=" {return XOR_ASSIGN;}
135 "|=" {return OR_ASSIGN;}
136
137 ">>" {return RIGHT_OP;}
138 "<<" {return LEFT_OP;}
139 "++" {return INC_OP;}
140 "--" {return DEC_OP;}
141 "->" {return PTR_OP;}
142 "&&" {return AND_OP;}
143 "||" {return OR_OP;}
144 "<=" {return LE_OP;}
145 ">=" {return GE_OP;}
146 "==" {return EQ_OP;}
147 "!=" {return NE_OP;}
148
149 "(" {return '(';}
150 ")" {return ')';}
151 ";" {return ';';}
152 ":" {return ':';}
153 "," {return ',';}
154 "=" {return '=';}
155 "." {return '.';}
156 "&" {return '&'|}
157 "!" {return '!'|}
158 "~" {return '~'|}
159 "-" {return '-'|}
160 "+" {return '+'|}
161 "*" {return '*'|}
162 "/" {return '/'|}
163 "%" {return '%'|}
164 "<" {return '<'|}
165 ">" {return '>'|}
166 "^" {return '^'|}
167 "|" {return '|'|}

```

```

168     "?;"    {return '?';}
169     ("{"|"<%") {return '{';}
170     ("}"|"%"") {return '}';}
171     ("["|"<:") {return '[';};
172     ("]"|">:") {return ']';}
173
174     {INCLUDE} {; }
175     [\t\n\f .]  {;};
176     . {}
177
178     /**
179
180     int yywrap(void) {
181         return 1;
182     }

```

## 2.2 Yacc

```

1  %{
2  #include <stdio.h>
// #define YYDEBUG 1
4  int ary[9] = {0,0,0,0,0,0,0,0,0};
5
6  // ary[] 배열에 가리킬 인덱스를 알기 쉽도록 정의
7  #define FUNC_CNT 0
8  #define OP_CNT 1
9  #define INT_CNT 2
10 #define CHAR_CNT 3
11 #define POINTER_CNT 4
12 #define ARRAY_CNT 5
13 #define SELECTION_CNT 6
14 #define LOOP_CNT 7
15 #define RETURN_CNT 8
16
17 // 반복문에서 else, else if 를 카운트 하지 않기 위해 사용하는 체크 변수
18 int isSelection = 0;
19
20 // int, char 타입 체크용도, int = 0, char = 1, 이외 타입에는 -1 을 저장
21 int nowType = -1;
22 int initVarCount = 0; // int a, b 와 같이 한번에 여러 변수를 생성할
→ 때의 변수의 개수
23
24 int isPointer = 0; // pointer 여부 체크
25 int isArray = 0;   // 배열 여부 체크
26
27 intisFunctionDeclaration = 0; // 함수 전방 선언 체크
28 int intParameterCount = 0;   // int 파라미터 개수 체크
29 int charParameterCount = 0;  // char 파라미터 개수 체크

```

```

30     int pointerParameterCount = 0; // pointer 파라미터 개수 체크
31
32     // typedef
33     char* identifier_name;           // lex에서 읽은 IDENTIFIER의 실제 이름
34     int isTypeDef = 0;              // typedef 여부 체크
35
36     char* type_name_array[1000];    // type name 심볼 테이블 역할 배열
37     int type_name_array_size = 0;
38
39     char* define_name_array[1000];  // #define 상수 심볼 테이블 역할 배열
40     int define_name_array_size = 0;
41
42     char* str_define_name_array[1000]; // #define 문자열 리터럴 심볼
43     ↪ 테이블 역할 배열
44     int str_define_name_array_size = 0;
45
46     int isPointerFunction = 0; // 함수 포인터 가능성 여부 체크
47
48     int yylex();
49
50     %token IDENTIFIER CONSTANT STRING_LITERAL SIZEOF DEFINE
51     %token PTR_OP INC_OP DEC_OP LEFT_OP RIGHT_OP LE_OP GE_OP EQ_OP NE_OP
52     %token AND_OP OR_OP MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN ADD_ASSIGN
53     %token SUB_ASSIGN LEFT_ASSIGN RIGHT_ASSIGN AND_ASSIGN
54     %token XOR_ASSIGN OR_ASSIGN TYPE_NAME
55
56     %token TYPEDEF EXTERN STATIC AUTO REGISTER
57     %token CHAR SHORT INT LONG SIGNED UNSIGNED FLOAT DOUBLE CONST
58     ↪ VOLATILE VOID
59     %token STRUCT UNION ENUM ELLIPSIS
60     %token CASE DEFAULT IF ELSE SWITCH WHILE DO FOR GOTO CONTINUE BREAK
61     ↪ RETURN
62
63     %start translation_unit
64
65     %%  

66     translation_unit
67     : external_declaration
68     | translation_unit external_declaration
69     ; // OK
70
71     external_declaration
72     : function_definition {
73         #if YYDEBUG
74             printf("\n함수 정의부 카운트 \n\n");
75         #endif
76
77         ary[FUNC_CNT]++;
78     }
79
80     %%
```

```

77     #if YYDEBUG
78         printf("int형 파라미터 %d 개\n", intParameterCount);
79     #endif
80     ary[INT_CNT] += intParameterCount;
81     intParameterCount = 0;
82
83     #if YYDEBUG
84         printf("char형 파라미터 %d 개\n", charParameterCount);
85     #endif
86     ary[CHAR_CNT] += charParameterCount;
87     charParameterCount = 0;
88
89     #if YYDEBUG
90         printf("pointer형 파라미터 %d 개\n",
91             → pointerParameterCount);
92     #endif
93     ary[POINTER_CNT] += pointerParameterCount;
94     pointerParameterCount = 0;
95 }
96 | declaration {
97     if (isFunctionDeclaration == 1) { // 함수 전방 선언은 무시한다.
98         #if YYDEBUG
99             printf("\n함수의 전방 선언은 무시합니다.\n");
100        #endif
101
102        intParameterCount = 0;
103        charParameterCount = 0;
104        pointerParameterCount = 0;
105    } else {
106        if (isPointer) {
107            ary[POINTER_CNT] += initVarCount;
108        }
109
110        initVarCount = 0;
111        isPointer = 0;
112        isFunctionDeclaration = 0;
113    }
114 | DEFINE IDENTIFIER CONSTANT { // "#define 식별자 상수" 처리
115     #if YYDEBUG
116         printf("add define : %s\n\n", identifier_name);
117     #endif
118     define_name_array[define_name_array_size++] = identifier_name;
119 }
120 | DEFINE IDENTIFIER STRING_LITERAL { // "#define 식별자 문자열" 처리
121     #if YYDEBUG
122         printf("add str define : %s\n\n", identifier_name);
123     #endif
124     str_define_name_array[str_define_name_array_size++] =
125         → identifier_name;

```

```

125 }
126 ; // OK
127
128 function_definition
129 : declaration_specifiers declarator declaration_list
130   ↪ compound_statement
131 | declaration_specifiers declarator compound_statement
132 | declarator declaration_list compound_statement
133 | declarator compound_statement
134 ; // OK
135
136 declaration
137 : declaration_specifiers ';' 
138 | declaration_specifiers init_declarator_list ';' {
139   if (isTypeDef) { // typedef 키워드가 등장했었다면, 새로운 타입에
140     ↪ 대한 선언이다.
141     #if YYDEBUG
142       printf("add type : %s\n\n", identifier_name);
143     #endif
144     type_name_array[type_name_array_size] = identifier_name;
145     type_name_array_size++;
146     isTypeDef = 0;
147   } else { // 아니라면 변수/함수에 대한 선언이다.
148     if (isPointer) {
149       ary[POINTER_CNT] += initVarCount;
150     }
151     if (nowType == 0) {
152       /* int 변수 카운트 */
153       #if YYDEBUG
154         printf("\nint 변수 선언: %d개\n\n", initVarCount);
155       #endif
156       ary[INT_CNT] += initVarCount;
157
158     } else if (nowType == 1) {
159       /* char 변수 카운트 */
160       #if YYDEBUG
161         printf("\nchar 변수 선언: %d개\n\n", initVarCount);
162       #endif
163       ary[CHAR_CNT] += initVarCount;
164     }
165   }
166   initVarCount = 0;
167   isPointer = 0;
168   nowType = -1;
169   isTypeDef = 0;
170 }
171 ;
172 init_declarator_list

```

```

173     : init_declarator                                { initVarCount++; }
174     | init_declarator_list ',' init_declarator    { initVarCount++; }
175     ;
176
177     init_declarator
178     : declarator
179     | declarator '=' initializer { ary[OP_CNT]++; }
180     ;
181
182     initializer
183     : assignment_expression
184     | '{' initializer_list '}'
185     | '{' init_declarator_list ',' '}'
186     ;
187
188     initializer_list
189     : initializer
190     | initializer_list ',' initializer
191
192     // IDENTIFIER //
193     identifier_list
194     : IDENTIFIER
195     | identifier_list ',' IDENTIFIER
196     ;
197
198     // TYPE //
199     typeSpecifier
200     : VOID
201     | CHAR {nowType = 1; /*char = 1*/}
202     | SHORT
203     | INT {nowType = 0; /* int = 0*/}
204     | LONG
205     | FLOAT
206     | DOUBLE
207     | SIGNED
208     | UNSIGNED
209     | struct_or_union_specifier
210     | enum_specifier
211     | TYPE_NAME
212     ;
213
214     typeQualifier
215     : CONST
216     | VOLATILE
217     ;
218
219     specifierQualifierList
220     : typeSpecifier specifierQualifierList
221     | typeSpecifier
222     | typeQualifier specifierQualifierList

```

```
223 | type_qualifier
224 ;
225
226 type_name
227 : specifier_qualifier_list
228 | specifier_qualifier_list abstract_declarator
229 ;
230
231 pointer
232 : '*' { isPointer = 1; }
233 | '*' type_qualifier_list
234 | '*' pointer
235 | '*' type_qualifier_list pointer
236 ;
237
238 type_qualifier_list
239 : type_qualifier
240 | type_qualifier_list type_qualifier
241 ;
242
243 // STRUCT //
244 struct_or_union_specifier
245 : struct_or_union IDENTIFIER '{' struct_declarator_list '}'
246 | struct_or_union '{' struct_declarator_list '}'
247 | struct_or_union IDENTIFIER
248 ;
249
250 struct_or_union
251 : STRUCT { nowType = -1; } // struct 타입이 등장하였으므로 int, char
252 ↵ 타입 체크 해제
253 | UNION
254 ;
255
256 struct_declarator_list
257 : struct_declarator
258 | struct_declarator_list struct_declarator
259 ;
260
261 struct_declarator
262 : specifier_qualifier_list struct_declarator_list ';' {
263     if (isPointer) {
264         ary[POINTER_CNT] += initVarCount;
265     }
266     if (nowType == 0) {
267         /* int 변수 카운트 */
268         #if YYDEBUG
269             printf("init int: %d\n", initVarCount);
270         #endif
271         ary[INT_CNT] += initVarCount;
272     } else if (nowType == 1) {
273 }
```

```

272     /* char 변수 카운트 */
273     #if YYDEBUG
274         printf("init char: %d\n\n", initVarCount);
275     #endif
276     ary[CHAR_CNT] += initVarCount;
277 }
278
279 initVarCount = 0;
280 isPointer = 0;
281 nowType = -1;
282 }
283 ;
284
285 struct_declarator_list
286 : struct_declarator {initVarCount++;}
287 | struct_declarator_list ',' struct_declarator {initVarCount++;}
288 ;
289
290 struct_declarator
291 : declarator
292 | ':' constant_expression
293 | declarator ':' constant_expression
294 ;
295
296 // OPERATOR //
297 unary_operator
298 : '&'
299 | '*'
300 | '+'
301 | '-'
302 | '^'
303 | '!'
304 ;
305
306 assignment_operator
307 : '='
308 | MUL_ASSIGN
309 | DIV_ASSIGN
310 | MOD_ASSIGN
311 | ADD_ASSIGN
312 | SUB_ASSIGN
313 | LEFT_ASSIGN
314 | RIGHT_ASSIGN
315 | AND_ASSIGN
316 | XOR_ASSIGN
317 | OR_ASSIGN
318 ;
319
320
321 // EXPRESSION //

```

```

322     expression
323         : assignment_expression
324         | expression ',' assignment_expression
325         ;
326
327     constant_expression
328         : conditional_expression
329         ;
330
331     assignment_expression
332         : conditional_expression
333         | unary_expression assignment_operator assignment_expression {
334             ↳ ary[OP_CNT]++;
335         }
336
337     conditional_expression
338         : logical_or_expression
339         | logical_or_expression '?' expression ':' conditional_expression
340         ;
341
342     cast_expression
343         : unary_expression
344         | '(' type_name ')' cast_expression { ary[OP_CNT]++; }
345         ;
346
347     unary_expression
348         : postfix_expression
349         | INC_OP unary_expression { ary[OP_CNT]++; }
350         | DEC_OP unary_expression { ary[OP_CNT]++; }
351         | unary_operator cast_expression
352         | SIZEOF unary_expression
353         | SIZEOF '(' type_name ')'
354         ;
355
356     postfix_expression
357         : primary_expression
358         | postfix_expression '[' expression ']'
359         | postfix_expression '(' ')'
360         | postfix_expression '(' argument_expression_list ')'
361         | postfix_expression '.' IDENTIFIER { ary[OP_CNT]++; }
362         | postfix_expression PTR_OP IDENTIFIER { ary[OP_CNT]++; }
363         | postfix_expression INC_OP { ary[OP_CNT]++; }
364         | postfix_expression DEC_OP { ary[OP_CNT]++; }
365
366     primary_expression
367         : IDENTIFIER
368         | CONSTANT
369         | STRING_LITERAL

```

```

369 | '(' expression ')'
370 ;
371
372 argument_expression_list
373 : assignment_expression
374 | argument_expression_list ',' assignment_expression
375 ;
376
377
378 // EXPRESSION WITH OPERATOR //
379 logical_or_expression
380 : logical_and_expression
381 | logical_or_expression OR_OP logical_and_expression {
382   ↪ ary[OP_CNT]++;
383 }
384 ;
385
386 logical_and_expression
387 : inclusive_or_expression
388 | logical_and_expression AND_OP inclusive_or_expression {
389   ↪ ary[OP_CNT]++;
390 }
391 ;
392
393 inclusive_or_expression
394 : exclusive_or_expression
395 | inclusive_or_expression '||' exclusive_or_expression {
396   ↪ ary[OP_CNT]++;
397 }
398 ;
399
400 exclusive_or_expression
401 : and_expression
402 | exclusive_or_expression '^' and_expression { ary[OP_CNT]++; }
403 ;
404
405 and_expression
406 : equality_expression
407 | and_expression '&' equality_expression { ary[OP_CNT]++; }
408 ;
409
410 equality_expression
411 : relational_expression
412 | equality_expression EQ_OP relational_expression { ary[OP_CNT]++; }
413 | equality_expression NE_OP relational_expression { ary[OP_CNT]++; }
414 ;
415
416 relational_expression
417 : shift_expression
418 | relational_expression '<' shift_expression { ary[OP_CNT]++; }
419 | relational_expression '>' shift_expression { ary[OP_CNT]++; }

```

```

414     | relational_expression LE_OP shift_expression { ary[OP_CNT]++; }
415     | relational_expression GE_OP shift_expression { ary[OP_CNT]++; }
416     ;
417
418 shift_expression
419   : additive_expression
420   | shift_expression LEFT_OP additive_expression { ary[OP_CNT]++; }
421   | shift_expression RIGHT_OP additive_expression { ary[OP_CNT]++; }
422   ;
423
424 additive_expression
425   : multiplicative_expression
426   | additive_expression '+' multiplicative_expression {
427     ↪ ary[OP_CNT]++;
428   | additive_expression '-' multiplicative_expression {
429     ↪ ary[OP_CNT]++;
430   ;
431
432 multiplicative_expression
433   : cast_expression
434   | multiplicative_expression '*' cast_expression { ary[OP_CNT]++; }
435   | multiplicative_expression '/' cast_expression { ary[OP_CNT]++; }
436   | multiplicative_expression '%' cast_expression { ary[OP_CNT]++; }
437   ;
438
439 // DECLARATOR //
440 declarator
441   : pointer direct_declarator { if (isArray) {ary[ARRAY_CNT]++;}
442     ↪ isArray = 0; }
443   | direct_declarator { if (isArray) {ary[ARRAY_CNT]++;}
444     ↪ isArray = 0; }
445   ;
446
447 direct_declarator
448   : IDENTIFIER
449   | '(' declarator ')' {
450     ↪ if (isPointer) {
451       ary[POINTER_CNT]++;
452       isPointer = 0;
453       isPointerFunction = 1; // 포인터 함수 선언의 가능성성이 있다.
454     } /* 무조건 변수를 나타낸다. */
455   }
456   | direct_declarator '[' constant_expression ']' { isArray = 1; }
457   | direct_declarator '[' ']' { isArray = 1; }
458   | direct_declarator '(' parameter_list ')' {
459     ↪ if (isPointerFunction == 1) {
460       ary[FUNC_CNT]++;
461       isPointerFunction = 0;
462       #if YYDEBUG

```

```

460         printf("\n함수 포인터 선언\n\n");
461     #endif
462 } else if (isFunctionDeclaration == 0) {
463     isFunctionDeclaration = 1;
464 }
465 }
466 | direct_declarator '(' identifier_list ')'
467     /*isFunctionDeclaration = 1;*/ } // ANSI C 문법, 테케에 없음.
468 | direct_declarator '(' ')'
469     { if
470     (isFunctionDeclaration == 0) isFunctionDeclaration = 1; }
471 ;
472
473 abstract_declarator
474 : pointer
475 | direct_abstract_declarator { if (isArray)
476     {ary[ARRAY_CNT]++; isArray = 0;} }
477 | pointer direct_abstract_declarator { if (isArray)
478     {ary[ARRAY_CNT]++; isArray = 0;} }
479 ;
480
481 direct_abstract_declarator
482 : '(' abstract_declarator ')'
483 | '[' ']' { isArray = 1; }
484 | '[' constant_expression ']' { isArray = 1; }
485 | direct_abstract_declarator '[' ']'
486 | direct_abstract_declarator '[' constant_expression ']'
487 | '(' ')'
488 | '(' parameter_type_list ')'
489 | direct_abstract_declarator '(' ')'
490 | direct_abstract_declarator '(' parameter_type_list ')'
491 ;
492
493 // PARAMETER //
494 parameter_declaration
495 : declaration_specifiers declarator {
496     if (nowType == 0) {
497         intParameterCount++;
498     } else if (nowType == 1) {
499         charParameterCount++;
500     }
501
502     if (isPointer == 1) {
503         pointerParameterCount++;
504     }
505
506     nowType = -1;
507     isPointer = 0;
508 }
509 | declaration_specifiers abstract_declarator {
510     #if YYDEBUG

```

```

506     printf("추상");
507 #endif
508     if (nowType == 0) {
509         intParameterCount++;
510     } else if (nowType == 1) {
511         charParameterCount++;
512     }
513
514     if (isPointer == 1) {
515         pointerParameterCount++;
516     }
517
518     nowType = -1;
519     isPointer = 0;
520 }
521 | declaration_specifiers /* 파라미터 변수는 아니고 선언만 있으면 세지
→ 않는다. */
522     if (nowType == 0) {
523         #if YYDEBUG
524             printf("\nint 형 파라미터 타입만 선언\n\n");
525         #endif
526     } else if (nowType == 1) {
527         #if YYDEBUG
528             printf("\nchar 형 파라미터 타입만 선언\n\n");
529         #endif
530     }
531
532     isPointer = 0;
533     nowType = -1;
534 }
535 ;
536
537 parameter_list
538 : parameter_declarator
539 | parameter_list ',' parameter_declarator
540 ;
541
542 parameter_type_list
543 : parameter_list
544 | parameter_list ',' ELLIPSIS
545 ;
546
547 // DECLARATION //
548 declaration_list
549 : declarator
550 | declaration_list declarator
551 ; // OK
552
553 storage_classSpecifier
554 : TYPEDEF { isTypeDef = 1; }

```

```

555 | EXTERN
556 | STATIC
557 | AUTO
558 | REGISTER
559 ;
560
561 declaration_specifiers
562 : storage_classSpecifier
563 | storage_classSpecifier declaration_specifiers { /* type def */}
564 | typeSpecifier { isPointer = 0; } // 타입이 나오면 기존 포인터
→ 체크는 변수에 쓰인 포인터가 아님
565 | typeSpecifier declaration_specifiers
566 | typeQualifier
567 | typeQualifier declaration_specifiers
568 ;
569
570 enum_specifier
571 : ENUM '{' enumerator_list '}'
572 | ENUM IDENTIFIER '{' enumerator_list '}'
573 | ENUM IDENTIFIER
574 ;
575
576 enumerator_list
577 : enumerator
578 | enumerator_list ',' enumerator
579 ;
580
581 enumerator
582 : IDENTIFIER
583 | IDENTIFIER '=' constant_expression { ary[OP_CNT]++; }
584 ;
585
586 // STATEMENT //
587 compound_statement
588 : '{' '}'
589 | '{' statement_list '}'
590 /* | '{' declaration_list '}' // declaration_statement // test
591 | '{' declaration_list statement_list '}' // 여기에서 둘 사이에
→ 순서를 없애주어야 함. */
592 ; // OK
593
594 statement_list
595 : statement { if (isSelection)
→ {ary[SELECTION_CNT]++; isSelection = 0; } }
596 | statement_list statement { if (isSelection)
→ {ary[SELECTION_CNT]++; isSelection = 0; } }
597 ; // OK
598
599 statement
600 : labeled_statement

```

```

601 | compound_statement
602 | declaration_or_expression_statement
603 //| expression_statement { printf("\n\nend line\n\n"); }
604 | selection_statement { isSelection = 1; }
605 | iteration_statement { ary[LOOP_CNT]++; }
606 | jump_statement
607 ;
608
609 labeled_statement
610 : IDENTIFIER ':' statement
611 | CASE constant_expression ':' statement
612 | DEFAULT ':' statement
613 ; // OK
614
615 declaration_or_expression_statement
616 : expression_statement
617 | declaration
618 ;
619
620 expression_statement
621 : ';'
622 | expression ';'
623 ; // OK
624
625 selection_statement
626 : IF '(' expression ')' statement
627 | IF '(' expression ')' statement ELSE statement
628 | SWITCH '(' expression ')' statement
629 ;
630
631 iteration_statement
632 : WHILE '(' expression ')' statement
633 | DO statement WHILE '(' expression ')' ';' '
634 //| FOR '(' expression_statement expression_statement ')'
635   ↪ statement
636 //| FOR '(' expression_statement expression_statement expression
637   ↪ ')' statement
638 | FOR '(' declaration_or_expression_statement expression_statement
639   ↪ ')' statement
640 | FOR '(' declaration_or_expression_statement expression_statement
641   ↪ expression ')' statement
642 ;
643
644 jump_statement
645 : GOTO IDENTIFIER ';'
646 | CONTINUE ';'
647 | BREAK ';'
648 | RETURN ';'           { ary[RETURN_CNT]++; }
649 | RETURN expression ';' { ary[RETURN_CNT]++; }
650 ; // OK

```

```
647 //  
648  
649  
650 int main(void)  
651 {  
652 #if YYDEBUG  
653     yydebug = 1;  
654 #endif  
655     yyparse();  
656     printf("function = %d\n", ary[0]);  
657     printf("operator = %d\n", ary[1]);  
658     printf("int = %d\n", ary[2]);  
659     printf("char = %d\n", ary[3]);  
660     printf("pointer = %d\n", ary[4]);  
661     printf("array = %d\n", ary[5]);  
662     printf("selection = %d\n", ary[6]);  
663     printf("loop = %d\n", ary[7]);  
664     printf("return = %d\n", ary[8]);  
665     return 0;  
666 }  
667  
668 void yyerror(const char *str)  
669 {  
670     fprintf(stderr, "error: %s\n", str);  
671 }
```