과제 #5. Yacc & Lex

소프트웨어학부 20180325 임성현

1. 과제 내용

- Yacc 명세와 Lex 명세를 완성하여 구문 분석기 프로그램을 제작 및 실험한다.

2. 문제 해결 방법

- 수업 강의 자료와 교재 부록을 활용하여 Yacc, Lex 명세를 작성한다.

- yyerror()와 yywrap()을 추가로 구현한다.

- yacc 동작 시, -d 옵션을 주어 y.tab.h 파일을 생성한다.

3. 실행 결과

1. 일반선언문

텍스트이(가) 표시된 사진

자동 생성된 설명

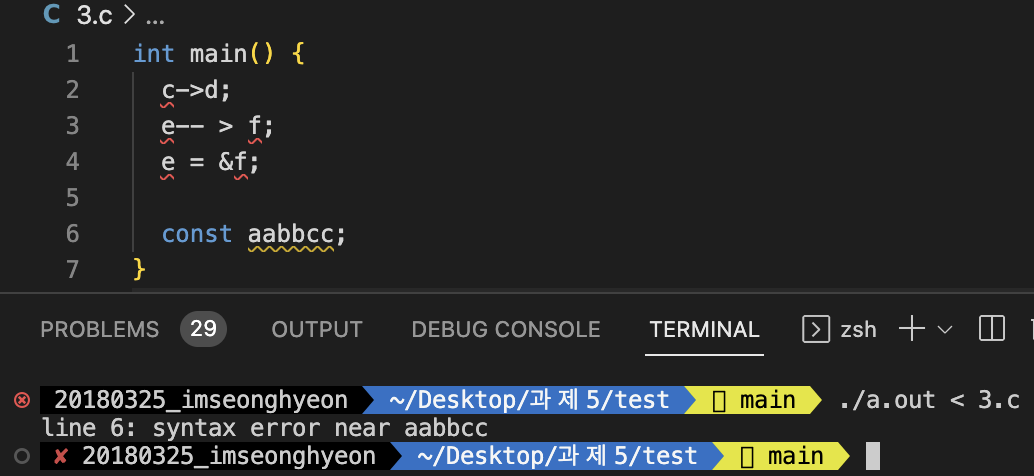
* int, float, char 타입의 선언이 정상적으로 이루어진다.
* 배열 타입의 선언이 정상적으로 이루어진다.
* storage\_class\_specifier (auto, static, typedef)의 선언이 정상적으로 이루어진다.
* struct, enum 타입의 선언이 정상적으로 이루어진다.
* double 타입의 선언은 에러가 난다. lex 명세 내, checkIdentifier() 에서 int, float, char, void 타입만 TYPE\_IDENTIFIER로 인식하게 했기 때문이다.

2. 포인터 및 배열 활용한 선언문과 각종 연산자

텍스트이(가) 표시된 사진

자동 생성된 설명

* int \* 배열과 float 함수 포인터 배열 선언이 정상적으로 이루어진다.
* 함수 선언이 정상적으로 이루어진다.
* 사칙연산 기호를 정상적으로 인식다.
* 전위 및 후위 증감 연산자를 정상적으로 인식한다.
* ‘\*=’ 와 같은 복합 대입 연산자는 인식이 에러가 난다. lex 명세 에서 언급되지 않았기 때문이다.



* 참조 연산자를 정상적으로 인식한다.
* const 키워드는 에러가 난다. . lex 명세에서 언급되지 않았기 때문이다.

3. 명령문

텍스트, 스크린샷, 모니터, 화면이(가) 표시된 사진

자동 생성된 설명

* switch – case 문을 정상적으로 인식한다. 단, case에는 정수가 와야 컴파일 에러가 나지 않으나 이는 semantic에 해당하는 얘기이니 syntax에 해당하지는 않는다.
* return을 정상적으로 인식한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* if – else if – else문을 정상적으로 인식한다.
* contiune, break를 정상적으로 인식한다.

텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명

- do – while 문을 정상적으로 인식한다.

- & 연산자를 정상적으로 인식한다.

텍스트, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명

* for 문을 정상적으로 인식하나, 해당 수업의 yacc 명세는 C89를 기반으로 작성되었으므로, C99 이후 버전부터 지원되는 for (int i=0;;)같은 문법에 에러가 발생한다.

텍스트, 모니터, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* for 문을 정상적으로 인식한다.
* ‘<<’ 와 같은 복합 대입 연산자는 인식이 에러가 난다. lex 명세 에서 언급되지 않았기 때문이다.

텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명

* sizeof 를 정상적으로 인식한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* ‘<=’, ‘>=’ 와 같은 비교 연산자를 정상적으로 인식한다.
* ‘&&’, ‘||’ 와 같은 논리 연산자를 정상적으로 인식한다.

4. 기타

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

* 3강 슬라이드 중, <어휘분석기의 확장>에서 언급 된 주석을 lex 명세에서 기술하지 않았다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* goto 문도 수업 내용 상, lex 명세에서 기술하지 않았다.

4. 결론

* 미리 기술한 yacc 명세와 lex 명세를 yacc과 lex 프로그램에 활용하여 구문 분석기 프로그램을 제작 및 실험하였다.
  + 선언문, 함수 정의, 명령문, 각종 수식을 정상적으로 분석한다.
  + yacc -d 옵션으로 y.tab.h 파일을 만들어 별도 define없이 yacc 명세에서 정의한 token들을 define하여 lex 명세에서 include “y.tab.h” 하는 방식이 있다.
* 사용하는 OS에 따라 yacc, lex 대신 byacc(berkerly), bison, flex 등의 버젼을 사용할 수 있다.

5. 문제점

* C89 문법을 기준으로 yacc 명세가 작성되었기 때문에 for(int i=0;;)같은 C89 이후 문법을 인식하지 못한다.
* lex 명세에서 언급되지 않은 연산자는 사용할 수 없다.
  + ‘\*=’, ‘+=’, ‘-=’, ‘/=’, ‘<<’, ‘>>’ 등
* lex 명세에서 언급되지 않은 키워드는 사용할 수 없다.
  + goto, const
  + checkIdentifier() 에서 double, string

6. 원시 프로그램 (소스코드)

**yacc.y**

%{

#include <stdio.h>

#include <stdlib.h>

int line\_no = 1;

%}

%token IDENTIFIER TYPE\_IDENTIFIER FLOAT\_CONSTANT INTEGER\_CONSTANT

%token CHARACTER\_CONSTANT STRING\_LITERAL PLUS MINUS PLUSPLUS

%token MINUSMINUS BAR AMP BARBAR AMPAMP ARROW

%token SEMICOLON LSS GTR LEQ GEQ EQL NEQ DOTDOTDOT

%token LP RP LB RB LR RR PERIOD COMMA EXCL STAR SLASH PERCENT ASSIGN

%token COLON AUTO\_SYM STATIC\_SYM TYPEDEF\_SYM

%token STRUCT\_SYM ENUM\_SYM SIZEOF\_SYM UNION\_SYM

%token IF\_SYM ELSE\_SYM WHILE\_SYM DO\_SYM FOR\_SYM CONTINUE\_SYM

%token BREAK\_SYM RETURN\_SYM SWITCH\_SYM CASE\_SYM DEFAULT\_SYM

%start program

%%

program

: translate\_unit

;

translate\_unit

: external\_declaration

| translate\_unit external\_declaration

;

external\_declaration

: function\_definition

| declaration

;

function\_definition

: declaration\_specifiers declarator compound\_statement

| declarator compound\_statement

;

declaration

: declaration\_specifiers SEMICOLON

| declaration\_specifiers init\_declarator\_list SEMICOLON

;

declaration\_specifiers

: type\_specifier

| storage\_class\_specifier

| type\_specifier declaration\_specifiers

| storage\_class\_specifier declaration\_specifiers

;

storage\_class\_specifier

: AUTO\_SYM

| STATIC\_SYM

| TYPEDEF\_SYM

;

init\_declarator\_list

: init\_declarator

| init\_declarator\_list COMMA init\_declarator

;

init\_declarator

: declarator

| declarator ASSIGN initializer

;

type\_specifier

: struct\_specifier

| enum\_specifier

| TYPE\_IDENTIFIER

;

struct\_specifier

: struct\_or\_union IDENTIFIER LR struct\_declaration\_list RR

| struct\_or\_union LR struct\_declaration\_list RR

| struct\_or\_union IDENTIFIER

;

struct\_or\_union

: STRUCT\_SYM

| UNION\_SYM

;

struct\_declaration\_list

: struct\_declaration

| struct\_declaration\_list struct\_declaration

;

struct\_declaration

: type\_specifier struct\_declarator\_list SEMICOLON

;

struct\_declarator\_list

: struct\_declarator

| struct\_declarator\_list COMMA struct\_declarator

;

struct\_declarator

: declarator

;

enum\_specifier

: ENUM\_SYM IDENTIFIER LR enumerator\_list RR

| ENUM\_SYM LR enumerator\_list RR

| ENUM\_SYM IDENTIFIER

;

enumerator\_list

: enumerator

| enumerator\_list COMMA enumerator

;

enumerator

: IDENTIFIER

| IDENTIFIER ASSIGN constant\_expression SEMICOLON

;

declarator

: pointer direct\_declarator

| direct\_declarator

;

pointer

: STAR

| STAR pointer

;

direct\_declarator

: IDENTIFIER

| LP declarator RP

| direct\_declarator LB constant\_expression\_opt RB

| direct\_declarator LP parameter\_type\_list\_opt RP

;

constant\_expression\_opt

: /\* empty \*/

| constant\_expression

;

parameter\_type\_list\_opt

: /\* empty \*/

| parameter\_type\_list

;

parameter\_type\_list

: parameter\_list

| parameter\_list COMMA DOTDOTDOT

;

parameter\_list

: parameter\_declaration

| parameter\_list COMMA 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

: LP abstract\_declarator RP

| LB constant\_expression\_opt RB

| LP parameter\_type\_list\_opt RP

| direct\_abstract\_declarator LB constant\_expression\_opt RB

| direct\_abstract\_declarator LP parameter\_type\_list\_opt RP

;

initializer

: constant\_expression

| LR initializer\_list RR

;

initializer\_list

: initializer

| initializer\_list COMMA initializer

;

statement

: labeled\_statement

| compound\_statement

| expression\_statement

| selection\_statement

| iteration\_statement

| jump\_statement

;

labeled\_statement

: CASE\_SYM constant\_expression COLON statement

| DEFAULT\_SYM COLON statement

| IDENTIFIER COLON statement

;

compound\_statement

: LR declaration\_list statement\_list RR

;

declaration\_list

: /\* empty \*/

| declaration\_list declaration

;

statement\_list

: /\* empty \*/

| statement\_list statement

;

expression\_statement

: SEMICOLON

| expression SEMICOLON

;

selection\_statement

: IF\_SYM LP expression RP statement

| IF\_SYM LP expression RP statement ELSE\_SYM statement

| SWITCH\_SYM LP expression RP statement

;

iteration\_statement

: WHILE\_SYM LP expression RP statement

| DO\_SYM statement WHILE\_SYM LP expression RP SEMICOLON

| FOR\_SYM LP expression\_opt SEMICOLON expression\_opt SEMICOLON expression\_opt RP statement

;

expression\_opt

: /\* empty \*/

| expression

;

jump\_statement

: RETURN\_SYM expression\_opt SEMICOLON

| CONTINUE\_SYM SEMICOLON

| BREAK\_SYM SEMICOLON

;

primary\_expression

: IDENTIFIER

| INTEGER\_CONSTANT

| FLOAT\_CONSTANT

| CHARACTER\_CONSTANT

| STRING\_LITERAL

| LP expression RP

;

postfix\_expression

: primary\_expression

| postfix\_expression LB expression RB

| postfix\_expression LP arg\_expression\_list\_opt RP

| postfix\_expression PERIOD IDENTIFIER

| postfix\_expression ARROW IDENTIFIER

| postfix\_expression PLUSPLUS

| postfix\_expression MINUSMINUS

;

arg\_expression\_list\_opt

: /\* empty \*/

| arg\_expression\_list

;

arg\_expression\_list

: assignment\_expression

| arg\_expression\_list COMMA assignment\_expression

;

unary\_expression

: postfix\_expression

| PLUSPLUS unary\_expression

| MINUSMINUS unary\_expression

| AMP cast\_expression

| STAR cast\_expression

| EXCL cast\_expression

| MINUS cast\_expression

| PLUS cast\_expression

| SIZEOF\_SYM unary\_expression

| SIZEOF\_SYM LP type\_name RP

;

cast\_expression

: unary\_expression

| LP type\_name RP cast\_expression

;

type\_name

: declaration\_specifiers

| declaration\_specifiers abstract\_declarator

;

multiplicative\_expression

: cast\_expression

| multiplicative\_expression STAR cast\_expression

| multiplicative\_expression SLASH cast\_expression

| multiplicative\_expression PERCENT cast\_expression

;

additive\_expression

: multiplicative\_expression

| additive\_expression PLUS multiplicative\_expression

| additive\_expression MINUS multiplicative\_expression

;

relational\_expression

: additive\_expression

| relational\_expression LSS additive\_expression

| relational\_expression GTR additive\_expression

| relational\_expression LEQ additive\_expression

| relational\_expression GEQ additive\_expression

;

equality\_expression

: relational\_expression

| equality\_expression EQL relational\_expression

| equality\_expression NEQ relational\_expression

;

logical\_and\_expression

: equality\_expression

| logical\_and\_expression AMPAMP equality\_expression

;

logical\_or\_expression

: logical\_and\_expression˚ƒ

| logical\_or\_expression BARBAR logical\_and\_expression

;

assignment\_expression

: logical\_or\_expression

| unary\_expression ASSIGN assignment\_expression

;

constant\_expression

: expression

;

expression

: assignment\_expression

;

%%

extern char \*yytext;

int yyerror(char \*s)

{

printf("line %d: %s near %s \n", line\_no, s, yytext);

exit(1);

}

int main()

{

yyparse();

printf("검사 완료\n");

}

int yywrap()

{

return(1);

}

**lex.l**

digit [0-9]

letter [a-zA-Z\_]

delim [ \t]

line [\n]

ws {delim}+

%{

#include "y.tab.h"

extern int line\_no;

int checkIdentifier(char \*);

%}

%%

{ws} { }

{line} {line\_no++;}

auto {return(AUTO\_SYM);}

break {return(BREAK\_SYM);}

case {return(CASE\_SYM);}

continue {return(CONTINUE\_SYM);}

default {return(DEFAULT\_SYM);}

do {return(DO\_SYM);}

else {return(ELSE\_SYM);}

enum {return(ENUM\_SYM);}

for {return(FOR\_SYM);}

if {return(IF\_SYM);}

return {return(RETURN\_SYM);}

sizeof {return(SIZEOF\_SYM);}

static {return(STATIC\_SYM);}

struct {return(STRUCT\_SYM);}

switch {return(SWITCH\_SYM);}

typedef {return(TYPEDEF\_SYM);}

union {return(UNION\_SYM);}

while {return(WHILE\_SYM);}

"++" {return(PLUSPLUS);}

"--" {return(MINUSMINUS);}

"->" {return(ARROW);}

"<" {return(LSS);}

">" {return(GTR);}

"<=" {return(LEQ);}

">=" {return(GEQ);}

"==" {return(EQL);}

"!=" {return(NEQ);}

"&&" {return(AMPAMP);}

"||" {return(BARBAR);}

"..." {return(DOTDOTDOT);}

"(" {return(LP);}

")" {return(RP);}

"[" {return(LB);}

"]" {return(RB);}

"{" {return(LR);}

"}" {return(RR);}

":" {return(COLON);}

"." {return(PERIOD);}

"," {return(COMMA);}

"!" {return(EXCL);}

"\*" {return(STAR);}

"/" {return(SLASH);}

"%" {return(PERCENT);}

"&" {return(AMP);}

";" {return(SEMICOLON);}

"+" {return(PLUS);}

"-" {return(MINUS);}

"=" {return(ASSIGN);}

{digit}+ {return(INTEGER\_CONSTANT);}

{digit}+\.{digit}+ {return(FLOAT\_CONSTANT);}

{letter}({letter}|{digit})\* {return(checkIdentifier(yytext));}

\"([^"\n]|\\["\n])\*\" {return(STRING\_LITERAL);}

\'([^'\n]|\'\')\' {return(CHARACTER\_CONSTANT);}

%%

int checkIdentifier(char \*s)

{

char \*table[] = {"int", "float", "char", "void"};

for (int i = 0; i < sizeof(table)/ sizeof(table[0]); i++) {

if (strcmp(table[i], s) == 0)

return(TYPE\_IDENTIFIER);

}

return(IDENTIFIER);

}