



**컴파일러\_과제 5**



|  |  |
| --- | --- |
| 과목명 | 컴파일러 |
| 교수명 | 유재우 교수님 |
| 학 과 | IT대학 컴퓨터학부 |
| 학 번 | 20201662 |
| 이 름 | 이정현 |
| 제출 일자 | 2022.10.10 |

**Ⅰ 개요**

본 과제의 목표는 yacc와 lex를 통해 간단한 C parser를 구현하는 것이다. 각 파일을 작성한 후, yacc -d yacc.y를 통해 y.tab.c를 생성하고 lex lex.l를 통해 lex.yy.c을 생성하였다. 이 때 y.tab.c가 존재해야지만 lex lex.l을 할 수 있기 때문에 yacc -d yacc.y를 먼저 실행하였다.

또한 C parser를 구현한 후, 제대로 된 parsing이 되는지 확인하기 위해 여러 가지 케이스를 만들어 이를 테스트해보았다.

**Ⅱ yacc.y 작성 및 y.tab.c 생성**

%{

#include <stdio.h>

#include <stdlib.h>

int line\_no = 1;

int yyerror(char \*s);

int yylex();

%}

%start program

%token IDENTIFIER TYPE\_IDENTIFIER CHARACTER\_CONSTANT STRING\_LITERAL FLOAT\_CONSTANT INTEGER\_CONSTANT ASSIGN MINUS PLUS SEMICOLON AMP

%token PERCENT SLASH STAR EXCL COMMA PERIOD COLON RR LR RB LB RP LP DOTDOTDOT BARBAR AMPAMP NEQ EQL GEQ LEQ GTR LSS ARROW MINUSMINUS AND XOR OR QUEST

%token PLUSPLUS WHILE\_SYM UNION\_SYM TYPEDEF\_SYM SWITCH\_SYM STRUCT\_SYM STATIC\_SYM SIZEOF\_SYM RETURN\_SYM IF\_SYM FOR\_SYM ENUM\_SYM ELSE\_SYM

%token DO\_SYM DEFAULT\_SYM CONTINUE\_SYM CASE\_SYM BREAK\_SYM AUTO\_SYM GOTO\_SYM

%token REGISTER\_SYM EXTERN\_SYM CONST\_SYM VOLATILE\_SYM

// selection\_statement의 모호성을 없애주기 위해 아래의 코드를 추가해주었다.

%nonassoc "then"

%nonassoc ELSE\_SYM

%%

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\_qualifier

| type\_specifier declaration\_specifiers

| storage\_class\_specifier declaration\_specifiers

| type\_qualifier declaration\_specifiers

;

storage\_class\_specifier

: AUTO\_SYM

| STATIC\_SYM

| TYPEDEF\_SYM

| REGISTER\_SYM

| EXTERN\_SYM

;

type\_qualifier

: CONST\_SYM

| VOLATILE\_SYM

;

init\_declarator\_list

: 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

: declarator

;

enum\_specifier

: ENUM\_SYM IDENTIFIER LR enumerator\_list RR

| ENUM\_SYM LR enumerator\_list RR

| ENUM\_SYM IDENTIFIER

;

enumerator\_list

: 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

: 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 %prec "then"

| 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

| GOTO\_SYM IDENTIFIER 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

;

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

;

and\_expression

: equality\_expression

| and\_expression AND equality\_expression

;

exclusive\_or\_expression

: and\_expression

| exclusive\_or\_expression XOR and\_expression

;

inclusive\_or\_expression

: exclusive\_or\_expression

| inclusive\_or\_expression OR exclusive\_or\_expression

;

logical\_and\_expression

: inclusive\_or\_expression

| logical\_and\_expression AMPAMP inclusive\_or\_expression

;

logical\_or\_expression

: logical\_and\_expression

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

;

conditional\_expression

: logical\_or\_expression

| logical\_or\_expression QUEST expression COLON conditional\_expression

;

constant\_expression

: expression

;

expression

: assignment\_expression

;

assignment\_expression

: conditional\_expression

| unary\_expression ASSIGN 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);

}

**Ⅲ lex.l 작성 및 lex.yy.c 생성**

digit [0-9]

letter [a-zA-Z\_]

delim [ \t]

line [\n]

ws {delim}+

%{

#include "y.tab.h"

#include <string.h>

extern int yylval;

extern int line\_no;

int makeString();

int checkIdentifier();

%}

%%

{ws} { }

{line} {line\_no++;}

auto {return(AUTO\_SYM);}

break {return(BREAK\_SYM);}

case {return(CASE\_SYM);}

const {return(CONST\_SYM);}

continue {return(CONTINUE\_SYM);}

default {return(DEFAULT\_SYM);}

do {return(DO\_SYM);}

else {return(ELSE\_SYM);}

enum {return(ENUM\_SYM);}

extern {return(EXTERN\_SYM);}

for {return(FOR\_SYM);}

goto {return(GOTO\_SYM);}

if {return(IF\_SYM);}

register {return(REGISTER\_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);}

volatile {return(VOLATILE\_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(EXCL);}

"\\*" {return(STAR);}

"\/" {return(SLASH);}

"\%" {return(PERCENT);}

"\&" {return(AMP);}

"\;" {return(SEMICOLON);}

"\+" {return(PLUS);}

"\-" {return(MINUS);}

"\=" {return(ASSIGN);}

"\&" {return(AND);}

"\^" {return(XOR);}

"\|" {return(OR);}

"\?" {return(QUEST);}

{digit}+ {yylval = atoi(yytext); return(INTEGER\_CONSTANT);}

{digit}\*\.{digit}\* {yylval = makeString(yytext); return(FLOAT\_CONSTANT);}

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

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

\'([^'\n]|\\\')\' {yylval = \*(yytext+1); return(CHARACTER\_CONSTANT);}

"\/\\*"([^\\*\/]|[^\\*]"\/"|"\\*"[^\/])\*"\\*\/" { }

"\/\/"[^\n]\* { }

%%

int makeString(char \*s)

{

char \*t;

t = malloc(strlen(s) + 1);

strcpy(t,s);

return((unsigned long int)t);

}

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);

}

makeString(s);

return(IDENTIFIER);

}

**Ⅳ 여러가지 case test**

에러 없이 검사가 끝나면 “검사 완료”가 출력된다.

- test1.c

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

- test2.c

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

-> 본 과제에서 콤마(,)는 제외하였기 때문에 콤마에 의해 에러가 발생한다.

- test3.c

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명



- test4.c

텍스트이(가) 표시된 사진

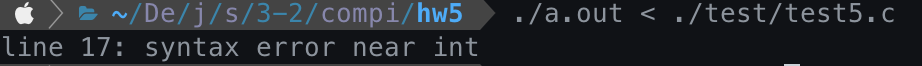
자동 생성된 설명



- test5.c

텍스트이(가) 표시된 사진

자동 생성된 설명

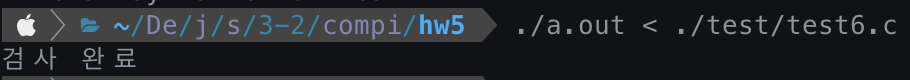


-> while (expression) statement에서 expression은 여러 가지 중 unary\_expression = assignment\_experession에 해당한다. 따라서 int가 unary\_expression으로 인식되었기 때문에 바로 다음에 ‘=’이 와야하지만 a가 왔기 때문에 에러가 발생한다.

- test6.c

텍스트이(가) 표시된 사진

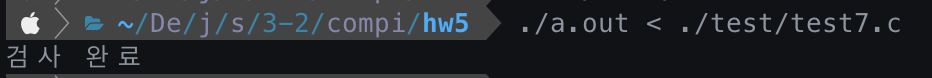
자동 생성된 설명



-test7.c

텍스트이(가) 표시된 사진

자동 생성된 설명



-test8.c

텍스트, 모니터, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 모니터, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

-test9.c

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

- test10.c

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명