ㄴㅡ

[Compiler]

과제 4번 : C 언어 Parser

25-1 유재우 교수님

20201740 소프트웨어학부 오승민

# Introduction

본 과제의 목표는 C언어의 문법 일부를 처리할 수 있는 구문 분석기(Parser) 를 제작하는 것이다. 이를 위해 Lex와 Yacc를 이용하여 각각 어휘 분석기와 구문 분석기를 구현하고, 이를 통해 C언어로 작성된 프로그램의 다양한 선언문과 명령문이 문법적으로 올바른지 검사할 수 있는 파서를 만든다.

이 파서는 C언어의 문법 요소 중 **표현식, 대입문, 조건식, 배열 및 구조체의 선언, ENUM 정의 등**을 처리할 수 있도록 설계되며, 내부적으로는 심볼 테이블과 타입 테이블을 구성하고, 이를 바탕으로 구문을 분석한 뒤, 필요에 따라 추상 구문 트리를 생성하도록 구현한다.

**1. Lex 프로그램 (oh.l)**

입력 파일에서 문자 단위로 읽어들여, C 언어의 키워드, 연산자, 식별자, 상수 등을 각각의 토큰으로 구분한다. yylex() 함수에서 인식된 토큰을 Yacc에 전달시키고, 소스 코드의 종료를 원만히 처리하기 위해 yywrap() 함수를 포함한다.

**2. Yacc 프로그램 (oh.y)**

C 언어의 선언문 및 명령문에 해당하는 문법 규칙을 정의하여 BNF 형식으로 작성한다. 각 문법 규칙에 대응되는 액션 코드를 정의하여 파싱 중 필요한 자료구조를 만들고, 구문 오류를 감지한다. 올바른 문법 구조를 만족하는 경우 파싱을 성공시키며, 오류가 발생하면 yyerror() 함수가 호출된다.

이러한 구성 요소들을 종합하여, 다양한 C언어의 문장들이 정상적으로 파싱 가능한지 실험하고, 그 결과를 통해 작성한 Lex / Yacc 프로그램의 문법 처리 능력을 검증하는 것이 본 과제의 핵심 목적이다.

파서가 처리할 수 있는 구문에는 단순한 대입문부터 구조체의 중첩 선언, 배열 초기화, 열거형 정의까지 포함되며, 실제 C 문법의 핵심적인 부분들을 충분히 커버할 수 있도록 설계되었다.

# Main Code

**oh.l**

digit [0-9]

letter [a-zA-Z]

delim [ \t]

line [\n]

ws {delim}+

%{

#include "oh.tab.h"

%}

%%

int { return(TYPE\_IDENTIFIER); }

char { return(TYPE\_IDENTIFIER); }

float { return(TYPE\_IDENTIFIER); }

void { return(TYPE\_IDENTIFIER); }

{ws} { }

{line} { }

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(TYPE\_IDENTIFIER); }

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(QUESTION); }

"\*" { 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(IDENTIFIER); }

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

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

%%

**oh.y**

%{

#include <stdio.h>

#include <stdlib.h>

extern char \*yytext;

int yyerror(char \*s);

int yylex();

int yywrap();

%}

%token IDENTIFIER TYPE\_IDENTIFIER FLOAT\_CONSTANT INTEGER\_CONSTANT STRING\_LITERAL CHARACTER\_CONSTANT

AUTO\_SYM BREAK\_SYM CASE\_SYM CONTINUE\_SYM DEFAULT\_SYM DO\_SYM ELSE\_SYM

ENUM\_SYM FOR\_SYM IF\_SYM RETURN\_SYM SIZEOF\_SYM STATIC\_SYM STRUCT\_SYM

SWITCH\_SYM TYPEDEF\_SYM UNION\_SYM WHILE\_SYM

PLUSPLUS MINUSMINUS ARROW LSS GTR LEQ GEQ EQL NEQ AMPAMP BARBAR DOTDOTDOT

LP RP LB RB LR RR COLON PERIOD COMMA EXCL STAR SLASH PERCENT AMP QUESTION SEMICOLON

PLUS MINUS ASSIGN

%start program

%%

program

: translation\_unit

;

translation\_unit

: external\_declaration

| translation\_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

;

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 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

: assignment\_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

;

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

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

;

constant\_expression

: expression

;

expression

: assignment\_expression

;

assignment\_expression

: logical\_or\_expression

| unary\_expression ASSIGN expression

| logical\_or\_expression QUESTION expression COLON assignment\_expression

;

%%

extern char \*yytext;

int main(int argc, char \*\*argv) {

yyparse();

printf("성공적으로 파싱을 마쳤습니다.\n프로그램을 종료합니다.\n");

}

int yyerror(char \*s) {

fprintf("에러 발생 : "%s , %s\n", s, yytext);

exit(1);

}

int yywrap() {

return (1);

}

기본적으로 "C언어와 컴파일러" 책에서 정의된 문법 규칙대로 작성한다.

추가적으로 수정한 부분은, 중괄호 초기화를 위해

initializer

: constant\_expression

| LR initializer RR

;

원래 constant\_expression으로 처리하던 저 부분을 assignment\_expression을 호출하는 것으로 바꾸어 중괄호 초기화 또한 문제 없이 해결한다.

initializer

: assignment\_expression

| LR initializer\_list RR

;

# Test Case

1. 기본 변수 선언

*int* main() {

*int* a;

*char* c;

*float* f;

return 0;

}

2. 변수 선언과 초기화

*int* main() {

*int* b = 10;

*char* d = 'x';

*float* pi = 3.14;

return 0;

}

3. 포인터 선언

*int* main() {

*int* \*ptr;

*char* \*name = "hello";

return 0;

}

4. 배열 선언

*int* main() {

*int* arr[5];

*char* str[10] = "world";

return 0;

}

5. 중첩 배열 선언 및 초기화

*int* main() {

*int* mat[2][3] = { {1, 2, 3}, {4, 5, 6} };

return 0;

}

6. 구조체 정의

*struct* Point {

*int* x;

*int* y;

};

7. 구조체 선언 + 변수

*struct* Point p1;

8. 구조체 초기화

*struct* Point {

*int* x;

*int* y;

};

*int* main() {

*struct* Point p2 = {1, 2};

return 0;

}

9. 공용체 선언

*union* Data {

*int* i;

*float* f;

*char* str[20];

};

10. 열거형 정의

*enum* Color {

RED,

GREEN,

BLUE

};

11. 열거헝과 값 지정

*enum* Weekday {

MON = 1, TUE, WED = 5

};

12. 함수 선언

*int* sum(*int* *a*, *int* *b*);

13. 함수 정의

*int* sum(*int* *a*, *int* *b*) {

return *a* + *b*;

}

*int* main() {

*int* result = sum(3, 4);

return 0;

}

14. if 문

*int* main() {

*int* x = 5;

if (x > 0)

x--;

else

x++;

return 0;

}

15. switch 문

*void* printDay(*int* *day*) {

switch(*day*) {

case 1: break;

case 2: break;

default: break;

}

}

16. while 반복문

*int* main() {

*int* n = 3;

while(n > 0) {

n--;

}

return 0;

}

17. do-while 반복문

*void* count(*int* *n*) {

do {

*n*--;

} while(*n* > 0);

}

18. for 반복문

*void* printNums() {

*int* i;

for(i = 0; i < 10; i++) {

;

}

}

19. 다양한 표현식

*int* main() {

*int* a = 1, b = 2;

*int* c = a + b \* 3 - 4 / 2;

*int* d = (a > b) ? a : b;

*int* e = a && b || c;

return e;

}

20. sizeof 연산자

*struct* Point {

*int* x;

*int* y;

};

*int* sizeTest() {

return sizeof(*int*) + sizeof(*struct* Point);

}

21. 복합 선언 리스트

*int* a, \*b, c[3], d = 5;

22. 함수 포인터

*int* func(*int*);

*int* (\*fptr)(*int*) = func;

23. 배열 안의 구조체

*struct* Point {

*int* x;

*int* y;

};

*struct* Point points[2] = { {1,2}, {3,4} };

24. 구조체 내부 포인터

*struct* Node {

*int* value;

*struct* Node \*next;

};

스크린샷, 패턴, 대칭, 다채로움이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

위의 Test Case를 모두 돌려본 결과, 파싱이 성공적임을 확인한다.