컴파일러 과제-4

20192800 권대현

1. 과제 내용

* 이번 컴파일러 과제는 2장의 문법과 3장에서 설명한 어휘 분석 프로그램을 토대로 C언어의 파서를 제작하여 C언어 코드의 문법을 검사하는 것이다.
* 입력으로 선언문, 명령문 및 함수가 포함된 프로그램들이 주어진다.
* 수식이 잘못된 경우 yyerror 함수를 통해 syntax error와 잘못된 yytext를 출력한다.
* 수식이 올바를 경우 success를 출력한다.

1. 해결 방법

* 먼저 프로그램 실행 환경은 cygwin으로 설정하였다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

* C언어 파서의 문법은 컴파일러-2장 강의노트.pdf 전체를, yacc 프로그램과 lex 문법은 컴파일러-3장 강의노트.pdf를 참고하여 작성했다.
* Lex와 yacc 명령어를 터미널에서 사용하기 위해 bison과 flex 패키지를 설치했다.
* 처음 yacc 프로그램을 돌렸을 때 중괄호{ }나 , 콤마를 잘못 사용하여 오류가 났었다. 이를 각각 LR RR, COMMA로 토큰명으로 바꿔줬다.
* 2장의 강의노트 pdf를 통해 yacc 문법을 작성할 때 콜론 : 문양을 잘못 이해하여 에러가 많이 떴었다. 이를 COLON으로 토큰명으로 고쳐서 해결했다.
* 이번 과제에서 type\_identifier를 구분 짓는 함수를 제외해야 했기 때문에 따로 int, float, void, char 형의 토큰을 추가하였다.
* 2개의 Shift/Reduce conflicts가 발생했다. 하나는 사전에 알고 있던 if – else ambiguity 문제이기에 넘어갔다.
* 다른 하나는 unary\_expression = assignment\_expression (SHIFT)와 unary\_expression -> constant\_expression = initializer (REDUCE)에 대한 상호 충돌이었다. 이 문제는 콤마 기호의 혼란을 없애기 위해 constant\_expression 쪽 문법과 initializer의 문법에서 expression을 assignment\_expression으로 변경해서 생긴 문제이다. 해당 conflicts를 해결하여 shift/reduce 개수를 줄여도 parser에서 제대로 parsing을 하지 않는 문제가 발생하였기에 그대로 두었다.
* 나머지는 reduce/reduce conflicts가 있다. Reduce/reduce의 경우 해결하기 위해 새로 예외 룰을 추가하거나 토큰 분류를 보다 상세히 해야 한다. 그러나 parser에서 문법의 오류를 검출하기엔 문제가 없음을 확인하여 그대로 두었다.
* 3장의 강의노트 pdf를 통해 lex 파일을 만들던 중 /\* \*/의 주석을 처리하는 정규식에 오류가 있음을 확인했다. 따라서 동일한 기능을 수행하는 정규식으로 코드를 수정했다.

1. 결론

* C언어 프로그램을 작성한 test.c와 완성된 parser 프로그램 a.exe를 터미널에서 실행한다. Test.c를 입력으로 넣어주기 위해 ./a.exe < test.c의 명령어를 입력하여 실행한다.
* Parser는 작성된 lex와 yacc 프로그램을 토대로 test.c의 문법을 검사한다. 만약 문법의 오류가 발견되면 parsing을 위해 임시로 저장했던 yytext를 출력하여 syntax error가 어느 부분에서 발생했는지를 알린다.
* C언어 프로그램에 문법적인 오류가 발생되지 않으면 success를 출력한다.
* 프로그램 실행결과
  + 올바른 프로그램
    - Test.c는 변수 declaration과 다양한 type들을 테스트했고, 이어서 function declaration을 테스트하였다. 각 function 안에는 여러 종류의 expression이 작성됐다.

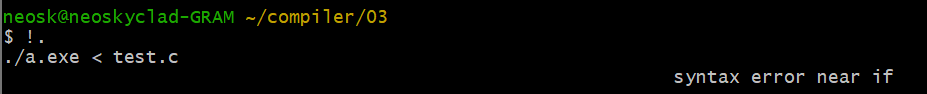
텍스트, 폰트, 스크린샷이(가) 표시된 사진

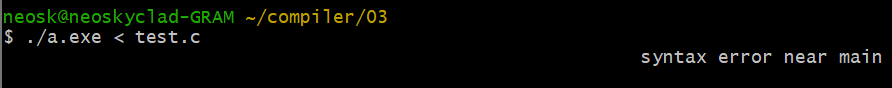
자동 생성된 설명

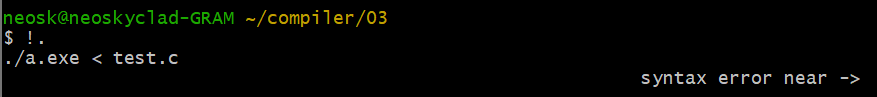
* + 잘못된 프로그램
    - Test.c에서 고의적으로 문법 오류를 내어 parser가 이를 감지하는지를 확인했다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명







1. 원시프로그램

* Lex

digit [0-9]

letter [a-zA-Z]

delim [\t]

line [\n]

ws {delim}+

%{

#include "y.tab.h"

%}

%%

{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 (TYPEDEF\_SYM); }

union { return (UNION\_SYM); }

while { return (WHILE\_SYM); }

goto { return (GOTO\_SYM); }

"\+\+" { return (PLUSPLUS); }

"\-\-" { return (MINUSMINUS); }

"\->" { return (ARROW); }

"<" { return (LSS); }

">" { return (GTR); }

"<=" { return (LEQ); }

">=" { return (GEQ); }

"==" { return (EQL); }

"!=" { return (NEQ); }

"&&" { return (AMPAMP); }

"||" { return (BARBAR); }

"<<" { return (LSH); }

">>" { return (RSH); }

"\.\.\." { 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); }

"\~" { return (NOT); }

"\^" { return (XOR); }

"\|" { return (BAR); }

"\?" { return (QUESTION); }

"const" { return (CONST\_SYM); }

"int" { return (INTEGER\_SYM); }

"float" { return (FLOAT\_SYM); }

"void" { return (VOID\_SYM); }

"char" { return (CHARACTER\_SYM); }

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

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

{letter}({letter}|{digit})\* { return (IDENTIFIER); }

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

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

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

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

%%

* Yacc

%token IDENTIFIER 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 GOTO\_SYM

PLUSPLUS MINUSMINUS ARROW LSS GTR LEQ GEQ EQL NEQ AMPAMP BARBAR LSH RSH DOTDOTDOT LP RP LB RB LR RR COLON PERIOD COMMA EXCL STAR SLASH PERCENT AMP SEMICOLON PLUS MINUS ASSIGN NOT XOR BAR QUESTION INTEGER\_CONSTANT FLOAT\_CONSTANT STRING\_LITERAL CHARACTER\_CONSTANT

CONST\_SYM INTEGER\_SYM FLOAT\_SYM VOID\_SYM CHARACTER\_SYM

%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 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

;

type\_qualifier : CONST\_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 : specifier\_qualifier\_list struct\_declarator\_list SEMICOLON

;

specifier\_qualifier\_list : type\_specifier

| type\_qualifier

| type\_specifier specifier\_qualifier\_list

| type\_qualifier specifier\_qualifier\_list

;

struct\_declarator\_list : struct\_declarator

| struct\_declarator\_list COMMA struct\_declarator

;

struct\_declarator : declarator

| constant\_expression

| declarator : constant\_expression

;

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

;

type\_identifier : INTEGER\_SYM | FLOAT\_SYM | VOID\_SYM | CHARACTER\_SYM

;

declarator : pointer direct\_declarator

| direct\_declarator

;

pointer : STAR type\_qualifier

| STAR type\_qualifier 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\_opt

;

abstract\_declarator\_opt : /\* empty \*/

| abstract\_declarator

;

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

| LR initializer\_list COMMA 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

| 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

| 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

| NOT 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

;

shift\_expression : additive\_expression

| shift\_expression LSH additive\_expression

| shift\_expression RSH additive\_expression

;

relational\_expression : shift\_expression

| relational\_expression LSS shift\_expression

| relational\_expression GTR shift\_expression

| relational\_expression LEQ shift\_expression

| relational\_expression GEQ shift\_expression

;

equality\_expression : relational\_expression

| equality\_expression EQL relational\_expression

| equality\_expression NEQ relational\_expression

;

AND\_expression : equality\_expression

| AND\_expression AMP equality\_expression

;

exclusive\_OR\_expression : AND\_expression

| exclusive\_OR\_expression XOR AND\_expression

;

inclusive\_OR\_expression : exclusive\_OR\_expression

| inclusive\_OR\_expression BAR 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 QUESTION expression COLON conditional\_expression

;

assignment\_expression : conditional\_expression

| unary\_expression ASSIGN assignment\_expression

;

comma\_expression : assignment\_expression

| comma\_expression COMMA assignment\_expression

;

expression : comma\_expression

;

constant\_expression : assignment\_expression

;

%%

#include <stdio.h>

#include <stdlib.h>

void main() { yyparse(); printf("success\n"); }

extern char \*yytext;

int yyerror(char \*s) { printf("%s near %s\n", s, yytext); exit(1); }

int yywrap() { return (1); }

* Test.c

//Declaration

int a,b = 10;

auto const int a;

static char \*\*c[10];

int \*p;

int a[10];

float \*f();

float (\*fun)();

//enum

typedef enum {false, true} BOOLEAN;

enum color {white, red = 10, green = 10+1, blue, black}

enum color c1, c2;

enum color {white, red, black} c1 = white;

//struct

typedef struct node {

    char \*name;

    int value, level;

    struct node \*link;

} NODE;

struct s1 { int b[3]; float c; } kim = { {1,2,3}, 4.5};

int \*a->b.c++;

int ++b = c++;

/\* Function Declaration \*/

main()

{

    return 0;

}

int func(int c)

{

    int i;

    switch(i)

    {

        case 0:

            break;

        case 1:

            continue;

        default:

            break;

    }

    if(a > b)

    {

        int t;

    }

    else

    {

        max = b;

    }

    return 1;

}

int fun (int a, float b, char c)

{

    static int x = 0;

    auto struct {int a; float b;} s;

    x = a \* a;

    result = result + x;

    while(a > b)

        max = a;

    for(i = 0; i < 100; i++)

    {

        max = b;

    }

    if(a << 1)

    {

        b >> 2;

    }

    else if (a && b || c)

    {

        sizeof(a);

    }

    else if (!b >= c)

    {

        printf("%d", ~c);

    }

    return x;

}