

컴파일러 과제-4

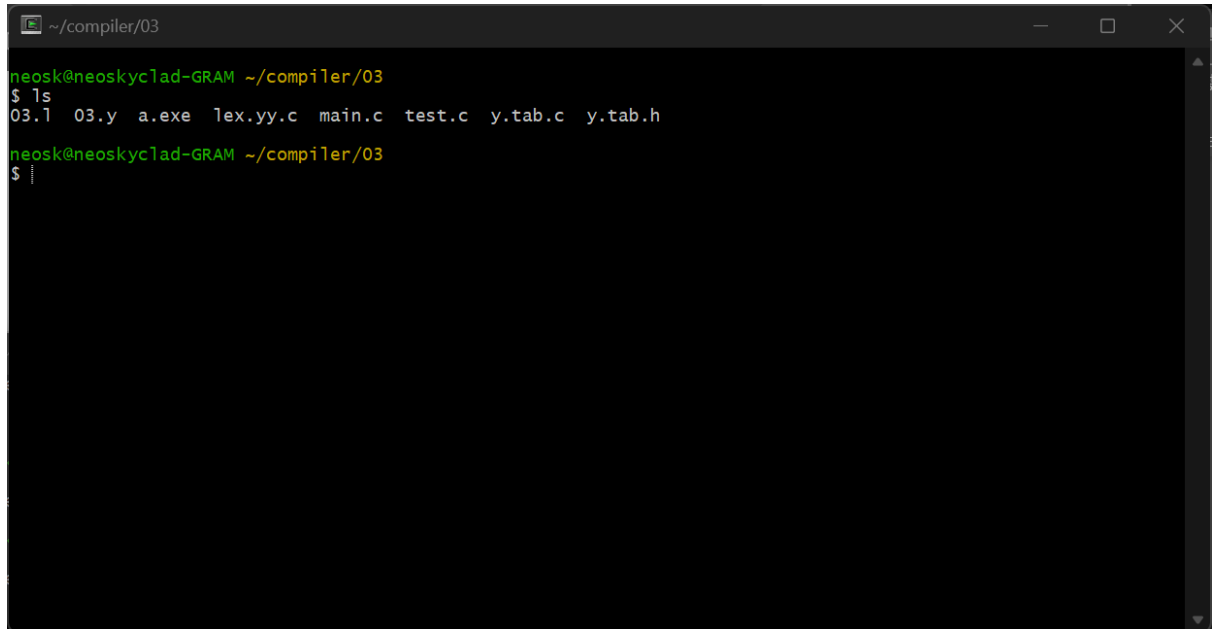
20192800 권대현

1. 과제 내용

- 이번 컴파일러 과제는 2장의 문법과 3장에서 설명한 어휘 분석 프로그램을 토대로 C 언어의 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 올바른 경우 success를 출력한다.

2. 해결 방법

- 먼저 프로그램 실행 환경은 cygwin으로 설정하였다.



```
~/compiler/03
neosk@neoscyclad-GRAM ~/compiler/03
$ ls
03.1 03.y a.exe lex.yy.c main.c test.c y.tab.c y.tab.h
neosk@neoscyclad-GRAM ~/compiler/03
$ |
```

- 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 났었다. 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.

3. 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.

```
neosk@neoskyclad-GRAM ~/compiler/03
$ !.
./a.exe < test.c

success
```

■ 잘못된 프로그램

- ◆ Test.c에서 고의적으로 문법 오류를 내어 parser가 이를 감지하는지를 확인했다.

```
neosk@neoskyclad-GRAM ~/compiler/03
$ !.
./a.exe < test.c

syntax error near =
```

```
neosk@neoskyclad-GRAM ~/compiler/03
$ !.
./a.exe < test.c

syntax error near if
```

```
neosk@neoskyclad-GRAM ~/compiler/03  
$ ./a.exe < test.c
```

syntax error near main

```
neosk@neoskyclad-GRAM ~/compiler/03  
$ !.  
./a.exe < test.c
```

syntax error near ->

4. 원시프로그램

- Lex

digit [0-9]

letter [a-zA-Z]

delim [Wt]

line [Wn]

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); }


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

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

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

"<" { return (LSS); }

">" { return (GTR); }

"<=" { return (LEQ); }

">=" { return (GEQ); }

"==" { return (EQL); }

"!=" { return (NEQ); }

"&&" { return (AMPAMP); }

"||" { return (BARBAR); }

"<<" { return (LSH); }

">>" { return (RSH); }

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

```

```
"W(" { return (LP); }

"W)" { return (RP); }

"W[" { return (LB); }

"W]" { return (RB); }

"W{" { return (LR); }

"W}" { return (RR); }

"W:" { return (COLON); }

"W." { return (PERIOD); }

"W," { return (COMMA); }

"W!" { return (EXCL); }

"W*" { return (STAR); }

"W/" { return (SLASH); }

"W%"      { return (PERCENT); }

"W&"      { return (AMP); }

"W;" { return (SEMICOLON); }

"W+"      { return (PLUS); }

"W-" { return (MINUS); }

"W="      { return (ASSIGN); }

"W~"      { return (NOT); }

"W^"      { return (XOR); }

"W|" { return (BAR); }

"W?" { 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}+W.{digit}+ { return (FLOAT_CONSTANT); }
```

```
{letter}({letter}){digit})* { return (IDENTIFIER); }
```

```
W"([ ^"Wn]|WW["Wn])*W" { return (STRING_LITERAL); }
```

```
W'([ ^'Wn]|W'W')W' { return (CHARACTER_CONSTANT); }
```

```
W/W*([ ^*]|W*+[ ^*/])*W*W/ { }
```

```
"/"/([ ^Wn)* { }
```

```
%%
```

- 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;  
}
```