1 - 1 yacc file

```
%{
   #include <stdio.h>

   extern char *yytext;
   extern int line_no;
   int syntax_err;

   void yyerror(char *s);
   int yylex();
   int yywrap();
%}

%start  program
%token
   IDENTIFIER TYPE_IDENTIFIER
   INTEGER_CONSTANT FLOAT_CONSTANT CHARACTER_CONSTANT
STRING_LITERAL
   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 SEMICOLON PLUS MINUS ASSIGN
   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

%%

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_list
   :
   | declaration_list declaration
   ;
```

```
declaration
    : declaration_specifiers SEMICOLON
    | declaration_specifiers init_declarator_list SEMICOLON
    ;
declaration_specifiers
    : type_specifier
    | storage_class_specifier
    | declaration_specifiers type_specifier
    | declaration_specifiers storage_class_specifier
    ;
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 enumarator_list RR
    | ENUM_SYM LR enumarator_list RR
    | ENUM_SYM IDENTIFIER
    ;
enumarator_list
    : enumarator
    | enumarator_list COMMA enumarator
    ;
enumarator
    : 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
    :
    | constant_expression
    ;
parameter_type_list_opt
    :
    | 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
    ;
compound_statement
    : LR declaration_list statement_list RR
    ;
statement_list
    :
    | 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
    :
    | 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
    :
    | 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
    ;

%%

int main() {
    line_no = 1;
    syntax_err = 0;

    yyparse();
    return 0;
}

void yyerror(char *s) {
    syntax_err++;
    printf("%s\n", s);
    printf("\tposition: %dline - near %s\n", line_no, yytext);
}

int yywrap() {
    return 1;
}
```

1 - 2 lex file

```
digit   [0-9]
letter  [a-zA-Z_]
delim    [ \t]
line    [\n]
ws      {delim}+

%{
    #include "y.tab.h"
    #include <stdio.h>

    extern int yylval;
    int line_no;

    char *makeString();
    int checkidentifier(char *s);
%}
```

```
%%
{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}+                    { 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); }

%%

char *makeString(char *s) {
    char *t;
    t = malloc(strlen(s) + 1);
    strcpy(t, s);
    return (t);
}

int checkidentifier(char *s) {
    char *table[] = {"int", "float", "char", "void"};
    int i = 0;

    for(; i < sizeof(table) / sizeof(char*); i++ ) {
        if(!strcmp(table[i], s)) return (TYPE_IDENTIFIER);
    }

    return (IDENTIFIER);
}
```

2 수행 결과

```
29
30      int
31
32      func1(a, b, c);
33
```

```
(base) somin@ubuntu:~/compiler$ ./a.out < test.c
syntax error
        position: 32line - near a
```

30 줄에 type_specifier 뒤에 SEMICOLON이 없기 때문에
컴파일러는 int func1(a, b, c); 즉 declaration으로 해석하는데
parameter_declaration에 declaration_specifiers가 없어서 a에서 오류라고 판단한 모습입니다.

```
34      for(x; x<a; x++) printf("test")
35      while(1) {break;}
36      do {x=1; continue;} while (a++);
37
```

```
(base) somin@ubuntu:~/compiler$ ./a.out < test.c
syntax error
        position: 35line - near while
```

34 줄에서 for () 다음 statement가 와야 하는데
printf(“test”) 는 SEMICOLON이 없기 때문에 statement 어디에도 속하지 않아서 오류라고 판단한 모습입니다.

```
42      1+2;
43      2-3;
44      3*4;
45      4/5;
46      !a;
47      ++b;
48      --c;
49      a < b;
50      b > a;
51      a <= b;
52      b >== a;
53      a == b;
54      a != b;
55      a && b;
56      a || b;
```

```
(base) somin@ubuntu:~/compiler$ ./a.out < test.c
syntax error
        position: 52line - near =
```

각종 expression들을 성공적으로 해석했고
52줄에 >== 처럼 틀린 부분에서 오류라고 판단하는 모습입니다.

```
56      a || b;
57      (int)a;
58      sizeof(int)
59  }
60
```

```
(base) somin@ubuntu:~/compiler$ ./a.out < test.c
syntax error
        position: 59line - near }
```

compound_statement 내에 declaration과 statement만 있어야 하는데
sizeof(int)는 SEMICOLON이 안찍혀서 아무것도 아니게 돼서 오류라고 판단하는 모습입니다.

```
1   struct node {
2       int a;
3   }
4
```

```
(base) somin@ubuntu:~/compiler$ ./a.out < test2.c
syntax error
        position: 4line - near
```

declaration_specifiers 뒤에 SEMICOLON이 오지 않아서 오류라고 판단하는 모습입니다.

- 따로 파일을 만든 이유 : 기존 테스트 파일이던 test.c에서 declaration_specifiers 뒤에 세미콜론을 안 붙여도 오류가 안납니다. 왜냐하면 그 뒤에 다른 declaration_specifiers나 init_declarator_list가 오기 때문입니다. 그래서 따로 파일을 만들어서 따로 테스트 해보았습니다. 문법적으로는 맞지만 의미상으로 틀리기 때문에 나중에 의미 분석하는 부분을 구현해야 오류라고 판단할 수 있습니다.

3 시행 착오 및 문제 해결 방법

- checkidentifier(char *s)
  이 함수에서 s가 type_identifier인지 아닌지 알아내야 합니다. 그래서 char*형 배열에 int, float, char, void (우리가 처리할 type_identifier들) 을 미리 넣어뒀습니다. 그리고 그 배열의 처음부터 끝까지 s랑 비교해보면서 s가 배열의 원소와 똑같은 게 있다면 type_identifier를 return 하였고, 똑같은 게 없다면 identifier를 return 하였습니다.

- 에러 핸들러
  yyerror(char *s)라고 에러를 처리하는 함수를 만들었습니다. yyerror는 생성된 y.tab.c에서 호출합니다. 오류가 발생하면 line_no (몇번째 줄에서 오류가 났는지)를 알려주고, 어떤 text 근처에서 오류가 났는지 (yytext) 출력합니다. 문제는 오류가 여러 개 있을 때에도 한개만 출력한다는 점인데, 이 점은 인터넷도 찾아보고 친구에게도 물어보는 등 몇일동안 고민해 보았으나 해결하지 못했습니다.

- declaration_specifier ; 에서 ;가 안와도 오류라고 하지 않는 점
  위에서 적었듯이 declaration_specifier 뒤에 세미콜론을 붙이지 않아도 오류라고 출력하지 않는 점에 대해서 오랫동안 고민해보고 오류라고 생각해서 고치려고

노력해보았으나 테스트 파일이 잘못된 것을 깨달았고 제 yyparse는 제대로 작성된 것이라는 것을 알았습니다.