MiniC 컴파일러 만들기(1): Lexical Analysis

컴파일러, 7번째 시간

경민기, 2025-10-28

컴파일러 구조

- 소스코드 → 어휘 분석 → 구문 분석 → 의미 분석 → 코드 생성
- 이번 주: 어휘 분석(Lexical Analysis)

어휘 분석이란?

- 입력: 문자열
- 출력: 토큰(token) 시퀀스
- 역할: 구문 분석기(parser)로 의미 단위 전달

```
int a = 3;

→ T_INT, T_ID(a), T_ASSIGN, T_NUM_INT(3), T_SEMI
```

FLex

- 입력: 정규표현식
- 출력: lex.yy.c
- 실행: 문자 입력 → 토큰 인식

lexer.l (1)

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

enum Token {
    T_INT, T_FLOAT, T_IF, T_ELSE, T_WHILE, T_RETURN,
    T_ID, T_NUM_INT, T_NUM_FLOAT,
    T_PLUS, T_MINUS, T_MUL, T_DIV,
    T_ASSIGN, T_EQ, T_NE, T_LT, T_LE, T_GT, T_GE,
    T_LPAREN, T_RPAREN, T_LBRACE, T_RBRACE, T_SEMI,
    T_UNKNOWN
};
```

lexer.l (2)

```
56   {WS}            ;     /* ignore whitespace */
57   "//".*          ;     /* line comment */
58   "/*"([^*]|\*+[^*/])*\*+ "/"   ; /* block comment */
59
60   "int"           { printf("T_INT\n"); }
61   "float"         { printf("T_FLOAT\n"); }
62   "if"            { printf("T_IF\n"); }
63   "else"          { printf("T_ELSE\n"); }
64   "while"         { printf("T_WHILE\n"); }
65   "return"        { printf("T_RETURN\n"); }
66
67   {FNUM}          { printf("T_NUM_FLOAT(%s)\n", yytext); }
68   {INUM}          { printf("T_NUM_INT(%s)\n", yytext); }
69   {ID}            { printf("T_ID(%s)\n", yytext); }
70
71   "=="            { printf("T_EQ\n"); }
72   "!="            { printf("T_NE\n"); }
73   "<="            { printf("T_LE\n"); }
74   ">="            { printf("T_GE\n"); }
75   "<"             { printf("T_LT\n"); }
76   ">"             { printf("T_GT\n"); }
77
78   "="             { printf("T_ASSIGN\n"); }
79   "+"             { printf("T_PLUS\n"); }
80   "-"             { printf("T_MINUS\n"); }
81   "*"             { printf("T_MUL\n"); }
82   "/"             { printf("T_DIV\n"); }
```

```
83
84   "("             { printf("T_LPAREN\n"); }
85   ")"             { printf("T_RPAREN\n"); }
86   "{"             { printf("T_LBRACE\n"); }
87   "}"             { printf("T_RBRACE\n"); }
88   ";"             { printf("T_SEMI\n"); }
89
90   .               { printf("T_UNKNOWN(%s)\n", yytext); }
91
92   %%
93   int main(void) {
94     yylex();
95     return 0;
96   }
97
```

테스트 코드

lexer.l | Makefile | sample.mc

9wk > minic > sample.mc

```
1    int a;
2    float x = 3.14;
3
4    a = a + 1;
5    if (a < 10) {
6      while (a != 0) {
7        a = a - 1;
8      }
9    }
10   return a;
11
```

빌드 & 테스트

lexer.l | Makefile | sample.mc

9wk > minic > sample.mc

```
1   int a;
2   float x = 3.14;
3
4   a = a + 1;
5   if (a < 10) {
6     while (a != 0) {
7       a = a - 1;
8     }
9   }
10  return a;
11
```

```
mingi_kyung@mg-tpx240:~/compiler_class/9wk/minic$ ./lexer < sample.mc
T_INT
T_ID(a)
T_SEMI
T_FLOAT
T_ID(x)
T_ASSIGN
T_NUM_FLOAT(3.14)
T_SEMI
T_ID(a)
T_ASSIGN
T_ID(a)
T_PLUS
T_NUM_INT(1)
T_SEMI
T_IF
T_LPAREN
T_ID(a)
T_LT
T_NUM_INT(10)
T_RPAREN
T_LBRACE
T_WHILE
T_LPAREN
T_ID(a)
T_NE
T_NUM_INT(0)
T_RPAREN
T_LBRACE
T_ID(a)
T_ASSIGN
T_ID(a)
T_MINUS
T_NUM_INT(1)
T_SEMI
T_RBRACE
T_RBRACE
T_RETURN
T_ID(a)
T_SEMI
```

다음 시간

- Bison으로 파싱 시작
- AST 구조로 확장

뱀다리: 다음 내용을 참고하세요.

하늘 아래 새로운 거 없습니다.

- https://github.com/SaiprakashShetty/C-Mini-Compiler-using-Lex-and-Yacc
- https://medium.com/codex/building-a-c-compiler-using-lex-and-yacc-446262056aaa

프로젝트 설계 시, 필요한 내용

- 1. 언어 개요 (Language Overview)
- 2. 데이터 타입 (Data Types)
- 3. 문법 구조 (Grammar Specification)
- 4. 제어 구조 (Control Structures)
- 5. 함수/스코프 규칙 (Functions & Scope)
- 6. 예제 프로그램 (Sample Programs)
- 7. 향후 확장 (Future Features)
- 8. 설계 이유 및 분석 (Design Rationale)