어휘분석 및 문맥자유문법의 활용

flex, bison 확장

경민기, 2025-09-30

DFA

- DFA = 상태와 전이 규칙으로 만들어진 토큰 인식기
  - Deterministic Finite Automaton = 유한 상태 기계의 한 종류
  - 입력 문자열을 한 글자씩 읽으면서 현재 상태를 바꾸고,
  - 마지막에 승인 상태(Accepting state)에 도달하면 해당 문자열을 인식함

DFA vs flex

DFA 요소	설명	flex에서의 대응
Σ (알파벳 집합)	DFA가 읽을 수 있는 입력 기호 집합	flex의 정규식 패턴에 등장하는 문자 집합예: [0-9], [a-zA-Z_]
Q (상태 집합)	DFA가 가질 수 있는 유한한 상태들	flex가 내부적으로 DFA 상태를 자동 생성 (개발자는 직접 상태를 정의할 필요 없음, 단 %x 등으로 명시적 상태 지정 가능)
q_0 (시작 상태)	DFA가 항상 여기서 시작	flex는 항상 “패턴 매칭 시작” 상태에서 실행됨
F (승인 상태 집합)	문자열을 인식 성공했을 때의 상태	flex의 규칙에서 **패턴이 매칭되면 곧바로 액션 실행** = 승인 상태
δ (전이 함수)	상태 + 입력 → 다음 상태	flex가 정규식을 기반으로 내부 DFA를 만들면서 δ를 정의함예: [0-9]+는 “숫자를 읽을 때마다 자기 자신으로 전이”

flex

- DFA를 자동화해주는 도구가 flex (Fast LEXical analyzer)
  - flex는 정의부 / 규칙부 / 사용자정의부로 나뉨
  - flex로 나눈 토큰은 bison으로 넘겨 처리

DFA로 토큰을 인식한다 → flex가 대신 해준다 → 우리가 할 일은 정규식 패턴 정의

문맥자유문법

- 문맥자유문법은 형식문법의 한 종류로, 규칙(생산 규칙, production rule)이 다음 형태를 가짐:
- $A \rightarrow \alpha$
  - A: 하나의 비단말 기호(non-terminal)
  - α: 단말 기호(terminal)와 비단말 기호의 문자열
- 예

  Expr → Expr + Term | Term
  Term → Term * Factor | Factor
  Factor → (Expr) | number

Bison

- CFG(문맥 자유 문법)을 기반으로 파서를 자동 생성하는 도구
- flex가 토큰을 넘겨주면, bison은 그것들을 문법 규칙에 따라 구조화
- flex와 마찬가지로 선언부/규칙부/사용자코드부로 나뉨

정리

- flex: 문자열을 읽어 토큰 반환
  - 예: 123 + 45 → NUMBER(123), PLUS, NUMBER(45)
- bison
  - 토큰 시퀀스를 문법에 맞게 분석
    - 예: expr → expr '+' expr 규칙 적용
  - semantic action 실행
    - $$ = $1 + $3; → 계산 결과 저장
  - 결과 반환 또는 AST 생성

실습 내용

- 정규식과 토큰 정의를 실제 스캐너 코드로 확장
- “문장 종결자”, “주석 제거”, “키워드/식별자 구분”을 구현
- (선택) 간단한 의미 검사/에러 처리까지 경험

Variable calc

% 연산자 추가하기

scanner.l

compiler_class > 5wk > variable_calc > scanner.l

```
1   %option noyywrap
2
3   %{
4   #include "parser.tab.h"
5   #include <stdlib.h>
6   #include <string.h>
7   %}
8
9   /* 패턴 */
10  ID      [A-Za-z_][A-Za-z0-9_]*
11  NUM     [0-9]+
12  WS      [ \t\r]+
13
14  %%
15
16  {ID}        { yylval.sval = strdup(yytext); return T_ID; }
17  {NUM}       { yylval.ival = atoi(yytext);   return T_NUMBER; }
18
19  "+"         { return '+'; }
20  "-"         { return '-'; }
21  "*"         { return '*'; }
22  "/"         { return '/'; }
23  "="         { return '='; }
24  "("         { return '('; }
25  ")"         { return ')'; }
26
27  \n          { return '\n'; }
28  {WS}        { /* skip */ }
29
30  .           { /* 알 수 없는 문자: 무시하거나 에러 처리 가능 */
31                /* fprintf(stderr, "unknown char: %s\n", yytext); */
32              }
33
34  %%
```

“%” { return ‘%’;}

Expression 추가하기

compiler_class > 5wk > variable_calc > parser.y

```
59  /* 줄 단위로 읽고, 줄마다 결과 출력 */
60  input
61    : /* empty */
62    | input stmt '\n'          { printf("%d\n", $2); }
63    | input '\n'               { /* 빈 줄 무시 */ }
64    ;
65
66  stmt
67    : expr                     { $$ = $1; }
68    | T_ID '=' expr            { setval($1, $3); $$ = $3; free($1); }
69    ;
70
71  expr
72    : expr '+' expr            { $$ = $1 + $3; }
73    | expr '-' expr            { $$ = $1 - $3; }
74    | expr '*' expr            { $$ = $1 * $3; }
75    | expr '/' expr            { if ($3 == 0) { yyerror("division by zero"); $$ = 0; } else $$ = $1 / $3; }
76    | '-' expr %prec UMINUS    { $$ = -$2; }
77    | '(' expr ')'             { $$ = $2; }
78    | T_NUMBER                 { $$ = $1; }
79    | T_ID                     { $$ = getval($1); free($1); }
80    ;
81
82  %%
```

```
| expr '/' expr          { if ($3 == 0) { yyerror("division by zero"); $$ = 0; } else $$ = $1 % $3; }
```

실행

문제 출력 디버그 콘솔 터미널 포트 직렬 모니터

```
mingi_kyung@mg-tpx240:~/compiler_class/5wk$ cd variable_calc/
mingi_kyung@mg-tpx240:~/compiler_class/5wk/variable_calc$ make
bison -d parser.y
parser.y: 경고 : 10 shift/reduce conflicts [-Wconflicts-sr]
parser.y: note: rerun with option '-Wcounterexamples' to generate conflict counterexamples
flex scanner.l
cc -Wall -g -o calc parser.tab.c lex.yy.c -lfl
lex.yy.c:1190:16: warning: ‘input’ defined but not used [-Wunused-function]
 1190 |     static int input  (void)
      |                ^~~~~
lex.yy.c:1147:17: warning: ‘yyunput’ defined but not used [-Wunused-function]
 1147 |     static void yyunput (int c, char * yy_bp )
      |                 ^~~~~~~
mingi_kyung@mg-tpx240:~/compiler_class/5wk/variable_calc$ ./calc
1+3
4
9%4
1
```

Malfunction clac

줄바꿈 제대로 하도록 수정하기

scanner.l

compiler_class > 5wk > ex01_basic_calc > scanner.l

```
1   /* scanner.l */
2   %option noyywrap
3
4   %{
5   #include "parser.tab.h"    /* bison이 생성하는 헤더 */
6   %}
7
8   DIGIT    [0-9]+
9   [redacted]
10
11  %%
12
13  {DIGIT}      { yylval.ival = atoi(yytext); return T_NUMBER; }
14  "+"          { return '+'; }
15  "*"          { return '*'; }
16  "("          { return '('; }
17  ")"          { return ')'; }
18  [redacted]
19
20  .            { /* 알 수 없는 문자 무시(혹은 에러 출력 가능) */ }
21
22  %%
```

\n   { return '\n'; }   /* 개행을 토큰으로 넘김 */

[\t\r]+ { /* 그냥 무시 */ }

parser.y

compiler_class > 5wk > malfunction_calc > parser.y

```
31  input
32      : /* empty */
33      | input expr                { printf("%d\n", $2); }
34      ;
35
36  expr
37      : expr '+' expr             { $$ = $1 + $3; }
38      | expr '*' expr             { $$ = $1 * $3; }
39      | '(' expr ')'              { $$ = $2; }
40      | T_NUMBER                  { $$ = $1; }
41      ;
42
43  %%
```

```
input
    : /* empty */
    | input expr '\n'       { printf("%d\n", $2); }
    | input '\n'            { /* 빈 줄 무시 */ }
    ;
```

실행

문제 출력 디버그 콘솔 터미널 포트 직렬 모니터

```
mingi_kyung@mg-tpx240:~/compiler_class/5wk/malfunction_calc$ make clean
rm -f calc lex.yy.c parser.tab.c parser.tab.h
mingi_kyung@mg-tpx240:~/compiler_class/5wk/malfunction_calc$ make
bison -d parser.y
flex scanner.l
cc -Wall -g -o calc parser.tab.c lex.yy.c -lfl
lex.yy.c:1156:16: warning: ‘input’ defined but not used [-Wunused-function]
 1156 |     static int input  (void)
      |                ^~~~~
lex.yy.c:1113:17: warning: ‘yyunput’ defined but not used [-Wunused-function]
 1113 |     static void yyunput (int c, char * yy_bp )
      |                 ^~~~~~~
mingi_kyung@mg-tpx240:~/compiler_class/5wk/malfunction_calc$ ./calc
2+3
5
4+123
127
```

Extended Calc

커맨드 추가, 연산자 추가

scanner.l

compiler_class > 5wk > extended_calc > scanner.l

```
12
13  %%
14
15  "print"     { return T_PRINT; }
16  {ID}        { yylval.sval = strdup(yytext); return T_ID; }
17
18  {NUM}       { yylval.ival = atoi(yytext);   return T_NUMBER; }
19
20  "+"         { return '+'; }
21  "-"         { return '-'; }
22  "*"         { return '*'; }
23  "%"         { return '%'; }
24  "="         { return '='; }
25  "("         { return '('; }
26  ")"         { return ')'; }
27
28  "//"[^\n]*  { /* skip */ }
29
30  {WS}        { /* skip */ }
31  \n          { return '\n'; }
32
33  .           { /* fprintf(stderr, "unknown char: %s\n", yytext); */ }
34
35  %%
```

parser.y (1)

compiler_class > 5wk > extended_calc > parser.y

```
59  /* 줄 단위로 즉시 reduce → 출력 */
60  input
61    : /* empty */
62    | input line
63    ;
64
65  line
66    : stmt '\n'              { printf("%d\n", $1); }
67    | '\n'                   { /* 빈 줄 무시 */ }
68    | error '\n'             { yyerrok; /* 에러 줄 스킵 */ }
69    ;
70
71  stmt
72    : expr                   { $$ = $1; }
73    | T_ID '=' expr          { setval($1, $3); $$ = $3; free($1); }
74    | T_PRINT expr           { $$ = $2; }  /* print도 값 출력(line에서 일괄 출력) */
75    ;
```

parser.y (2)

```
77  expr
78    : expr '+' expr          { $$ = $1 + $3; }
79    | expr '-' expr          { $$ = $1 - $3; }
80    | expr '*' expr          { $$ = $1 * $3; }
81    | expr '%' expr          { if ($3 == 0) { yyerror("mod by zero");       $$ = 0; } else $$ = $1 % $3; }
82    | '-' expr %prec UMINUS  { $$ = -$2; }
83    | '(' expr ')'           { $$ = $2; }
84    | T_NUMBER               { $$ = $1; }
85    | T_ID                   { $$ = getval($1); free($1); }
86    ;
87
88  %%
```

과제

과제 내용: 계산기 만들기

- 계산기에서 다음 기능을 지원 (4)
  - 사칙연산: + - * /
  - 괄호: (...)
  - 변수 대입
  - 줄 단위 실행: 엔터(\n) 또는 세미콜론(;)으로 문장 끝 구분

- 추가 기능 (+1)
  - 정수 -> 실수
  - print 문을 변형하여 show 문을 만들기
  - 에러처리