[Bottom Parsing을 이용한 Mini-C 인터프리터 개발]

학과: 소프트웨어학과
학번: 201520980
이름: 정진선

1. 서론

C언어에 기반한 간단한 Interpreter 언어 “MINI-C”를 위한 Interpreter이다. 본 Interpreter는 두 단계로 실행된다.
1단계: 단위(문장 또는 함수)별로 코드를 syntax tree로 변경
2단계: syntax tree를 이용하여 실행한 후 그 결과를 출력

1.1 상수

상수는 **2가지(int, double)** 형태를 지원한다.

1.2 변수

변수는 선언하지 않고 사용된다. 즉, 사용할 시 symbol table에 자동 추가되며, 초기화 시키지 않을 시 **0으로 초기화** 된다. (Local이 아닌 변수는 전역변수이므로)

변수의 type은 저장되는 값에 의해 결정되어야 하지만 본 interpreter에서는 int 와 double 모두 double로 저장되도록 했다.

1.3 수식(expression과 연산자

- 가감승제 연산자(+, -, *, /)
- 비교 연산 (>, >=, <, <=, ==, !=)
- assignment 연산자 (=)
- 정수, 실수 혼합식은 실수로 형을 변환하도록 했다.

1.4 문장

-수식문장: 수식에 ‘;’를 넣어 문장을 만든다.
-if 문 : if (exp) statement else statement
- while 문: while (exp) statement
- *출력문*: print exp; // exp의 계산 결과를 화면에 출력함

[Block 문 {statement_list}, 함수는 구현하지 않았다]

2. 문제 분석

- grammar rule

stat_list: /*empty */ | stat_list stat
stat:expr ; | print_stat |control_stat
print_stat: PRINT expr ';'
control_stat: if_stat | while_stat
if_stat: IF '(' expr ')' stat ELSE stat
while_stat:WHILE '(' expr ')' stat
expr: value
        |variable
        |variable ASSIGN expr
        |expr PLUS expr
        |expr MINUS expr
        |expr MUL expr
        |expr DIV expr
        |expr GT expr
        |expr GE expr
        |expr LT expr
        |expr LE expr
        |expr EE expr
        |expr NE expr
        |'(' expr ')'
value: INT | REAL
variable: ID

- Shift/Reduce Conflict, Reduce/Reduce Conflict등은 연산자 우선순위, 좌/우측 결합을 이용하여 해결했다.

좌측 결합
        LEFT: PLUS MINUS
        LEFT: MUL DIV
        LEFT: GT GE LT LE EE NE

우측 결합
        RIGHT: ASSIGN

우선순위: PLUS,MINUS < MUL, DIV

3. 설계

- 주요 자료구조(syntax tree 노드, symbol table 등)

TOKEN TYPES

```
%token PLUS(+) MINUS(-) MUL(*) DIV(/) GT(>) GE(>=) LT( <) LE(<=) EE(==) NE(!=)
%token ASSIGN (=)
%token IF(if) ELSE(else) PRINT(print) WHILE(while)
```

lex에서 토큰을 파싱할 때 그 형태를 분류하기 위한 token 값.
기타 '(', ')' ';' 등은 token을 허용했지만 그 외 토큰 값들은 lexical error로 간주했다.

Symbol Table

```
struct ST{
        char symbol[SYMBOL_MAX+1];
        TOKEN type;
        union {
                int integer_constant;
                double real_constant;
        } value;
};
```

symbol
- 변수일 경우 이름을 저장하도록 함

type
- 노드가 INT,DOUBLE,ID 등등을 구별하도록 하는 토큰 값

value
- Int token일 경우 integer_constant에, double일 경우 real_constant에 저장하기 위한 union.
하지만 실제론 real_constant만 사용함.

Syntax Node

```
struct SN{
    char symbol[SYMBOL_MAX+1];
    union{
        double real_constant;
        int integer_constant;
    }value;
    TOKEN token;
    struct SN* expr; //조건문 넣는곳
    struct SN* Left;
    struct SN* Right;
};
```

symbol, value, token 값은 symbol table 과 동일

expr: 비교 문장할 때 조건문을 넣는 곳이다.
Ex)

- if (**expr**) stat else stat
- while(**expr**) stat

Left, Right:

ASSIGN의 경우 variable을 left에 expr을 right에 할당했다.
if문의 경우 if (expr) left else right 으로 할당했다.
while문의 경우 while(expr) left 로 할당했다.

4. 수행 결과

1. 정상출력
2. LEXICAL ERROR (! 등을 추가한 것)
3. Syntax ERROR (문법에 맞지 않은 문장 추가. ex) ;;)

```
jinsun@jinsun-900X3K:~/Desktop/컴파일러/compiler_project/LEXYACCMINIC$ ./minic < sample.mc
20.000000
jinsun@jinsun-900X3K:~/Desktop/컴파일러/compiler_project/LEXYACCMINIC$ ./minic < sample_lexicalerror.mc
lexical error
jinsun@jinsun-900X3K:~/Desktop/컴파일러/compiler_project/LEXYACCMINIC$ ./minic < sample_syntaxerror.mc
syntax error
```