[컴파일러 – 수식 인터프리터 개발]

과제#2

학번: 201520980

학과: 소프트웨어학과

이름: 정진선

1. 서론

1.1 과제

Recursive Descent Parsing 기법을 이용하여 간단한 expression을 위한 인터프리터를 개발한다. expression은 아래와 같이 constants, variables, binary operators, unary operators, 지정 수식 (assignment expression)을 포함하며 괄호를 허용한다.

[Operator]

Binary Operator : +, -, * , / , =

Unary Operator : -

[상수]

-Integer

-Real Number

[Assignment Operator]

“=” : Right Association

구현된 부분:

1. Sam Lexical Analyzer 활용
2. Recursive Descent Parser
3. Syntax Tree 생성
4. Evaluator (숫자, 가감승제, Assignment Expression)

구현되지 않은 부분:

Integer 와 Real-Number에 출력에 있어서 구분하지 않음(모두 Real – Number 형식으로 출력. 소수점 아래 6자리)

2. 문제분석

2.1 Grammar Rule 분석

A → id A' | F' T' E'

A' → = A | T' E'

E → TE'

E' → + T E‘ | ε

T → F T'

T' → * F T' | ε

F → id | F'

F' → (E) | inum | fnum | - F

2.2 Recursive Descent Parsing을 이용한 수식 계산기의 기본 개념정리

GRAMMAR	FUNCTION
A()	1.다음 TOKEN이 ID일 경우, ID NODE “call”생성 후, restA(call) 반환. 2. 다음 TOKEN이 ID가 아닐 경우, restE(restT(restF())) 반환
restA(call)	1. 다음 TOKEN이 ASSIGN일 경우, ASSIGN NODE(Left = call, Right = A()) 생성 후 반환 2. 다음 TOKEN이 ASSIGN이 아닐경우 restE(restT(call)) 반환
E()	restE(T())반환
restE(call)	1. 다음 TOKEN이 PLUS일 경우, PLUS NODE(Left = call, Right=restE(T())) 생성 후 반환 2. 다음 TOKEN이 MINUS일 경우, MINUS NODE(Left = call, Right=restE(T())) 생성 후 반환 3. 다음 TOKEN이 PLUS, MINUS가 모두 아닐 경우, call 반환
T()	restT(F()) 반환
restT(call)	1. 다음 TOKEN이 MUL일 경우, MUL NODE(Left = call, Right = restT(F())) 생성 후 반환. 2. 다음 TOKEN이 DIV일 경우, DIV NODE(Left = call, Right = restT(F())) 생성 후 반환. 3. 다음 TOKEN이 MUL, DIV 모두 아닐 경우, call 반환
F()	1. 다음 TOKEN이 ID일 경우, ID NODE 생성 후 반환. 2. 다음 TOKEN이 ID가 아닐 경우, restF() 반환
restF()	1. 다음 TOKEN이 LP 일 경우, RP 검사 후, E() 반환 2. 다음 TOKEN이 INT 일 경우, INT NODE 생성 후 반환 3. 다음 TOKEN이 REAL일 경우, REAL NODE 생성 후 반환. 4. 다음 TOKEN이 MINUS일 경우, MINUS NODE(Left = (int)0, Right = F()) 생성 후 반환. 5. 다음 TOKEN이 1,2,3,4 모두 아닐 경우, 에러 표시 후 NULL NODE 반환.

3. 설계

3.1 정의된 구조체

TOKEN_LIST
+ TOKEN: token
+ char[]: value[TOKEN_VALUE_MAX+1]

SYMBOL_TABLE
+ char[]: symbol[SYMBOL_MAX+1]
+ TOKEN: token
+ int, double: value

SYNTAX_NODE
+ char[]: symbol[SYMBOL_MAX+1]
+ TOKEN: token
+ SYNTAX_NODE*: Left, Right

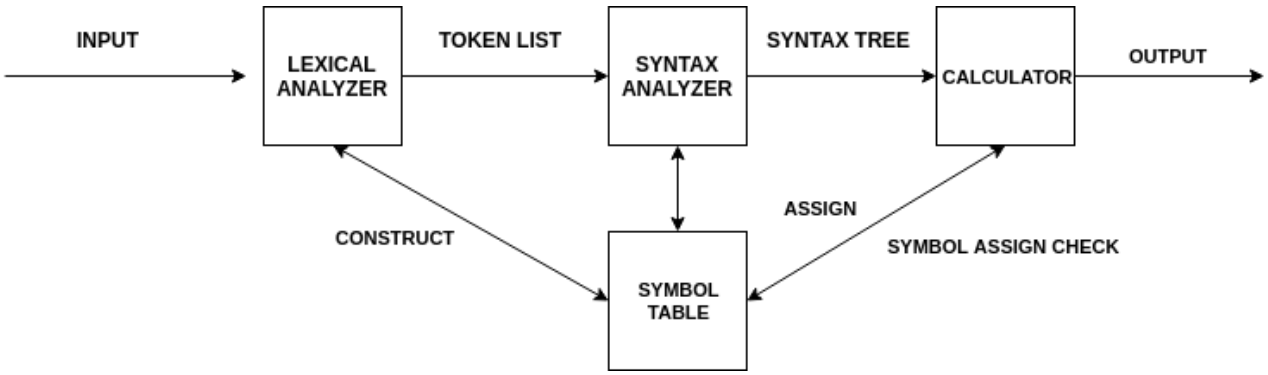
TOKEN
ENUM: [ID = 1, INT, REAL, PLUS, MINUS, MUL, DIV, ASSIGN, LP, RP]

TOKEN LIST: TOKEN 배열

SYMBOL TABLE: SYMBOL 기호와 값 유지

SYNTAX_NODE: 수식 SYNTAX TREE의 각 NODE. 이진트리를 형태로 CHILD가 LEFT,RIGHT 존재

3.2 LEXICAL ANALYZER, SYNTAX ANALYZER(SYNTAX TREE, CALCULATOR)관계



LEXICAL ANANYZER : INPUT을 입력받아, TOKEN LIST, SYMBOL TABLE 생성

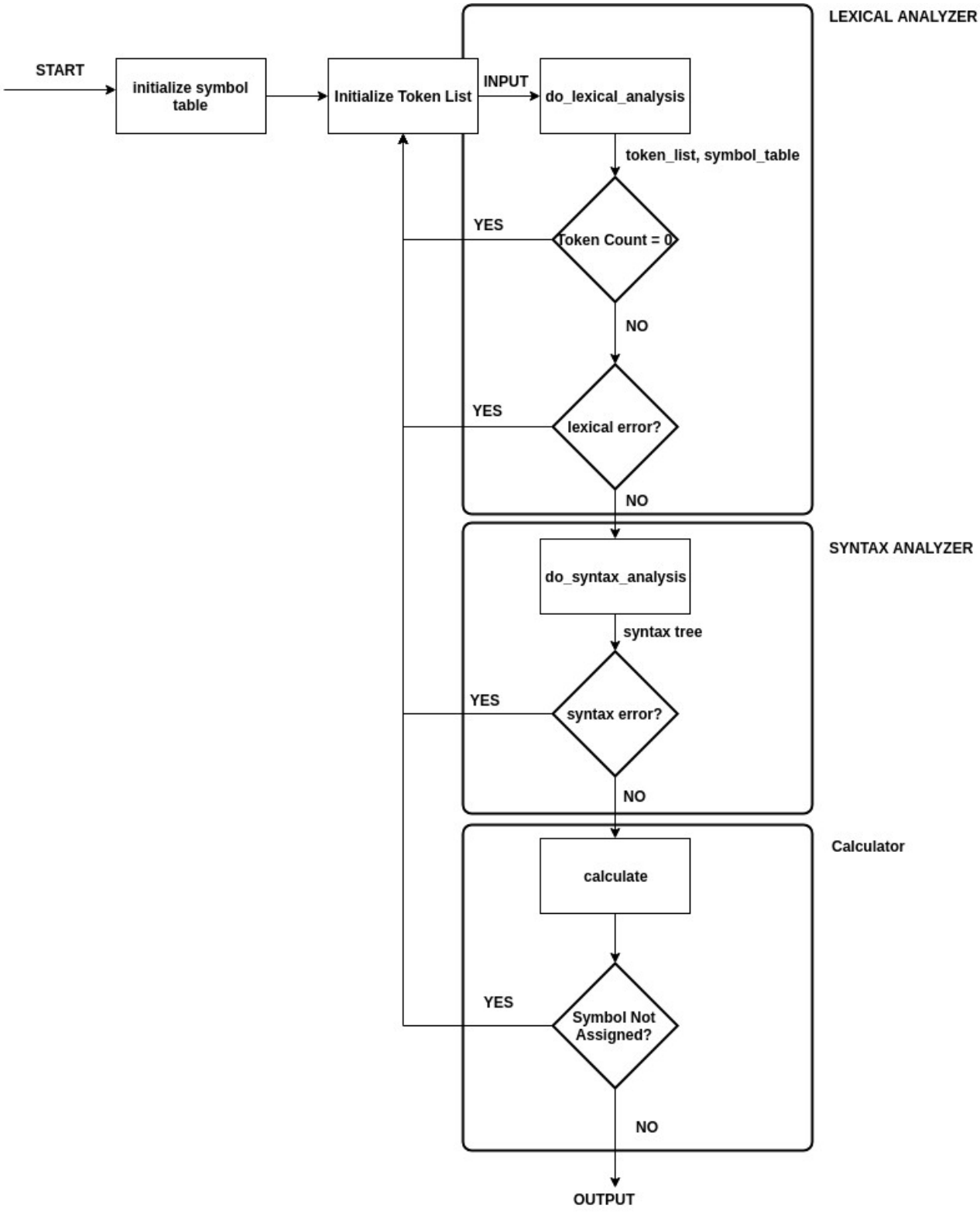
SYNTAX ANALYZER : TOKEN LIST를 참조하여 GRAMMAR 문법에 맞는지 체크하면서 SYNTAX TREE 생성

CALCULATOR : SYNTAX TREE로 가감승제 계산 후 출력

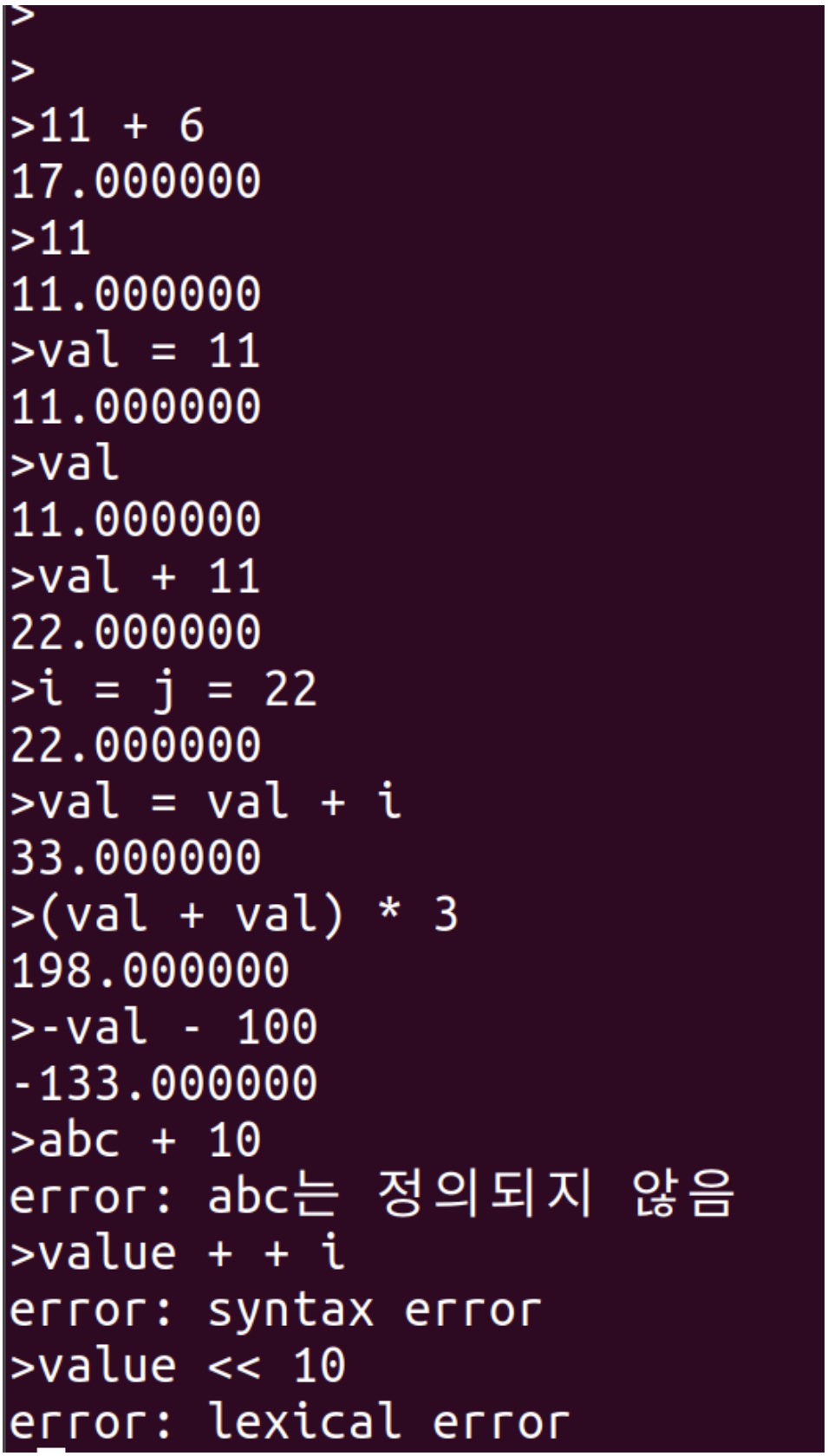
<"Unary Operation '-'의 경우, Binary Operation와 동일하게 작동하도록 Left Child에 0을 추가했습니다">

ex) -3 ==> 0-3, -a ==> 0-a 와 동일하게 인식

3.4 FLOW CHART



4. 수행 결과


>
>
>11 + 6
17.000000
>11
11.000000
>val = 11
11.000000
>val
11.000000
>val + 11
22.000000
>i = j = 22
22.000000
>val = val + i
33.000000
>(val + val) * 3
198.000000
>-val - 100
-133.000000
>abc + 10
error: abc는 정의되지 않음
>value + + i
error: syntax error
>value << 10
error: lexical error