Internal Document-20226XX2 신성섭

1. 문법

<program> → <statements>
<statements>→ <statement> | <statement><semi_colon><statements>
<statement> → <ident><assignment_op><expression>
<expression> → <term><term_tail>
<term_tail> → <add_op><term><term_tail> | ε
<term> → <factor> <factor_tail>
<factor_tail> → <mult_op><factor><factor_tail> | ε
<factor> → <left_paren><expression><right_paren> | <ident> | <const>
<const> → any decimal numbers
<ident> → any names conforming to C identifier rules
<assignment_op> → :=
<semi_colon> → ;
<add_operator> → + | -
<mult_operator> → * | /
<left_paren> → (
<right_paren> →)

* const의 경우 양의 정수로 한정

2. 처리 가능한 오류들

1) statements -> <statement> {; <statement>} 에서 ;가 누락 된 경우 “(Warning) ; 입력 누락. ; 입력 추가”를 띄우고 ;를 추가해 준다.

2) <ident> 에서 사용자가 작성한 식별자 이름이 C 식별자 이름에 위배 하는 경우 “(Error) C언어에 적합하지 않은 식별자 이름.(‘ident 이름’) 식별자 이름 변경 필요!!” 를 띄운다. 이때 프로그램 문법에서는 잘못된 식별자지만, 파싱 과정 중에는 정상적인 식별자 이름이라고 가정하고, 식별자 개수랑 식별자의 값에서 식별자인 것처럼 취급한다.

3) <statement> -> <ident> := <expression> 에서 :=이 : 나 = 로 작성 된 경우 "(Warning) (: | =)로 입력 됨. := 입력으로 변경" 을 띄우고 :=로 수정 해준다.

4) <statement> -> <ident> := <expression> 에서 ident 다음에 := 없는 경우 “(Warning) 문법 위배, :=입력 추가” 을 띄우고 :=를 추가한다.

5) <statement> -> <ident> := <expression>에서 A:= ; 와 같은 구조로 expression이 아예 작성 되지 않고 바로 ;가 오는 경우 “(Warning) 대입 연산자 뒤에 expression이 없음. 0을 추가 시킴. (‘ident’)은 0으로 초기화”를 띄우고 A;=0; 인 것 처럼 취급한다.

6) <expression> -> <term> {(+|-) <term>} 에서 A:=+3;과 같은 경우 “(Warning) iden:=(+|-) 숫자 구조. iden:= 0 (+|-) 숫자 구조로 변경.”을 띄우고 A:=0+3;으로 변경 한다.

7) <expression> -> <term> {(+|-) <term>} 에서 (+|-) 다음에 연달아 (+|-) 오는 경우 부호 논리에 따라 +나 - 로 바꿔 준다. (중복 연산자 제거)
부호논리: ++ 인 경우 + , +-인 경우 - , -+인 경우 -. --인 경우 +
부호논리에 따라 “(Warning) 중복 연산자(+) 발생 ++ 이므로 +로 변경” 와 같은 형식으로 오류 메세지를 출력한다.

8)<expression> -> <term> {(+|-) <term>} 에서 +- 다음에 아무 피연산자가 나오 지 않고 바로 ;가 나오는 경우 0을 자동 추가 해주고 “(Warning) (+|-) 뒤에 피연산자 미 입력. 0 자동 추가” 를 띄운다. 예를 들면 A:=3+; 인 경우 A:=3+0;으로 바꿔 준다.

9) <term> -> <factor> {(*|/) <factor>} 에서 A:=*4*3; 과 같은 경우 "(Warning) 잘못된 (*|/) 를 삽입. 삭제함"을 띄우고 A:=4*3;으로 바꿔준다.

10) <term> -> <factor> {(*|/) <factor>} 에서 (*|/) 다음에 피연산자 없이 바로 ;가 오는 경우 1을 자동 추가 해준다. 예를 들어 A:=3*; 와 같은 경우 "(Warning) (*|/) 뒤에 피연산자가 입력되지 않음. 1을 자동 추가 ₩n" 을 띄우고 A:=3*1;로 수정해 준다.

11) <term> -> <factor> {(*|/) <factor>} 에서 나누기의 경우 두번 째 피연산자가 0 인 경우 zero division error를 발생시키고 Unknown을 반환한다. 예를 들어 A:=3/0 과 같은 경우 “(Error) 0으로 나누기 시도. 결과값은 Unknown으로 반환”을 띄우고 Unknown을 반환해 최종 적으로 A는 Unknown이 된다.

12) <term> -> <factor> {(*|/) <factor>}에서 (*|/) 다음에 또 (*|/)가 나오는 경우 맨 앞의 연산자만을 이용하고 나머지 연산자는 무시한다. 예를 들어 A:=3*/2; 와 같은 경우 A:=3*2;와 같이 처리한다. 이때 (Warning) 잘못된 (*|/) 의 삽입. (*|/)를 삭제함을 띄운다.

13) 식별자에서 값을 가져오는 과정에서 식별자의 값이 정의 된 적 없으면 “(Error) 정의되지 않은 변수(‘변수명’)이 참조 됨. Unknown 반환”을 띄운다.

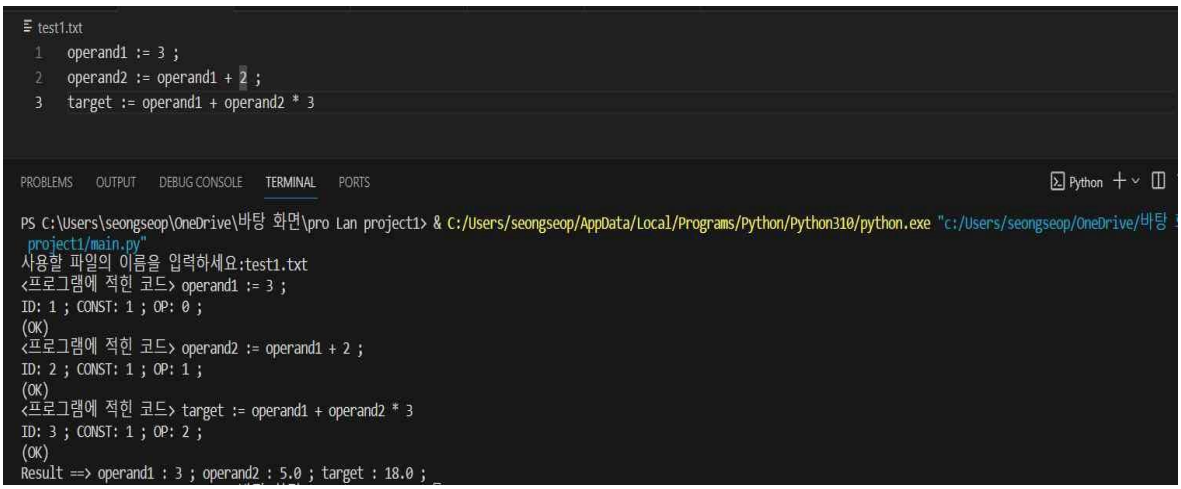
14) <factor> -> (<expression>) 에서) 괄호가 나오지 않는 경우 “(Warning)) 가 안 닫힘.) 추가.”를 띄운다고)를 추가해 준다. 예를 들면 A:=(1+3 ; 과 같은 경우 A:=(1+3);로 수정해 준다.

15) INT_literal 앞에 +가 있는 경우 불필요한 + 이므로 “(Warning) + 이므로 + 삭제”를 띄운다. - 인 경우 “(Warning) 상수로 음수를 이용”을 띄운다.

16) 위 오류를 제외한 나머지 문법 오류 들은 "(Error) 허용되지 않은 문법 입력. Unknown 반환₩n" 을 출력하고 Unknown을 반환한다.

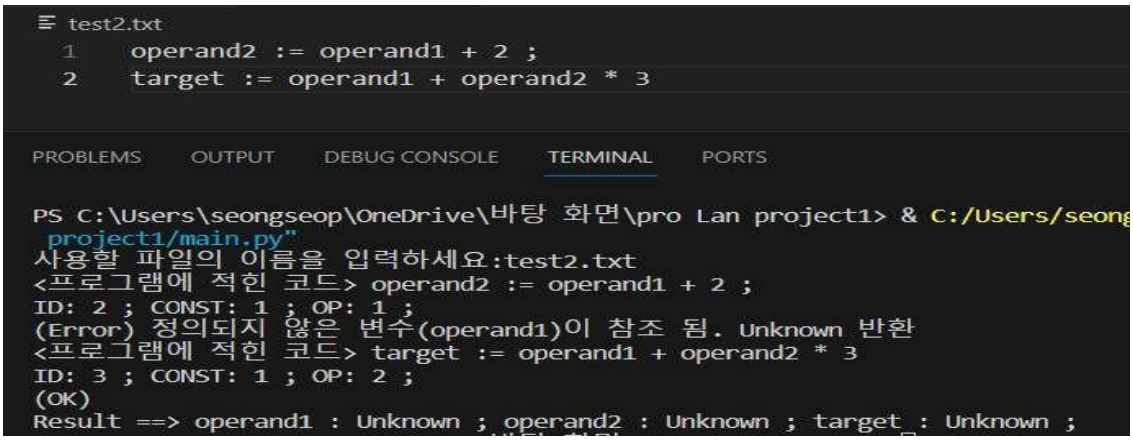
3. 파싱 실행 예시

1) test1 파일. (예시1과 동일)



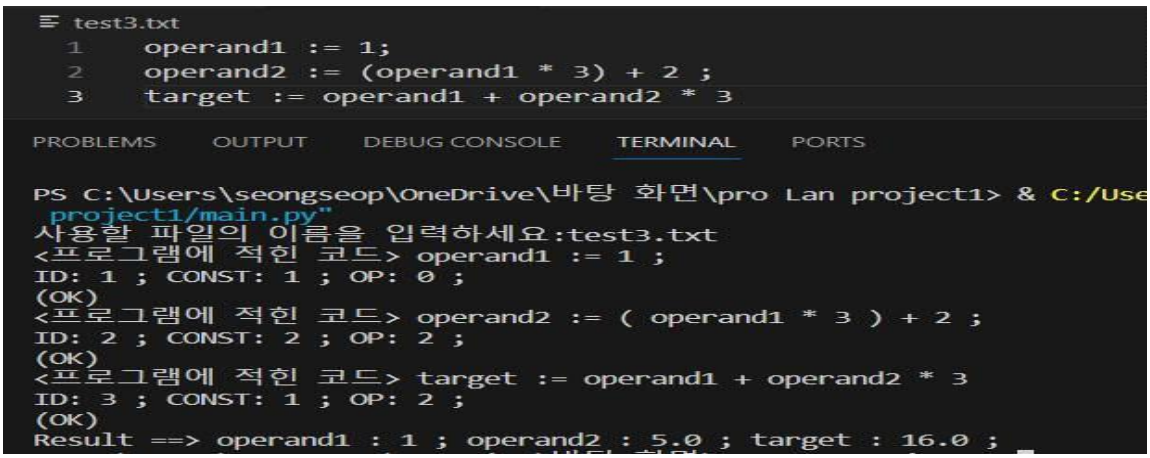
-오류가 없는 경우

2) test2 파일. (예시2과 동일)



- 에러 13번(정의되지 않는 변수 사용) 발생

3) test3 파일. (예시3과 동일)



-오류가 없는 경우

4) test4 파일. (예시4과 동일)

```
test4.txt
1    operand1 := 3 ;
2    operand2 := operand1 + + 2 ;
3    target := operand1 + operand2 * 3

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\seongseop\OneDrive\바탕 화면\pro Lan project1> & C:/Users/seongs
 project1/main.py"
사용할 파일의 이름을 입력하세요:test4.txt
<프로그램에 적힌 코드> operand1 := 3 ;
ID: 1 ; CONST: 1 ; OP: 0 ;
(OK)
<프로그램에 적힌 코드> operand2 := operand1 + + 2 ;
<수정한 코드> operand2 := operand1 + 2 ;
ID: 2 ; CONST: 1 ; OP: 1 ;
(Warning) 중복 연산자(+) 발생 ++ 이므로 +로 변경
<프로그램에 적힌 코드> target := operand1 + operand2 * 3
ID: 3 ; CONST: 1 ; OP: 2 ;
(OK)
Result ==> operand1 : 3 ; operand2 : 5.0 ; target : 18.0 ;
```

-에러 7번(++ 사용) 발생

5) test5 파일.

```
test5.txt
1    1operand1 := 3 ;
2    operand2 := operand1 + + 2
3    target := operand1 + operand2 * 3

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\seongseop\OneDrive\바탕 화면\pro Lan project1> & C:/Users/seongseop/AppData/L
 project1/main.py"
사용할 파일의 이름을 입력하세요:test5.txt
<프로그램에 적힌 코드> 1operand1 := 3 ;
ID: 1 ; CONST: 1 ; OP: 0 ;
(Error) c언어에 적합하지 않은 식별자 이름.(1operand1) 식별자 이름 변경 필요!!
<프로그램에 적힌 코드> operand2 := operand1 + + 2
<수정한 코드> operand2 := operand1 + 2 ;
ID: 2 ; CONST: 1 ; OP: 1 ;
(Error) 정의되지 않은 변수(operand1)이 참조 됨. Unknown 반환
(Warning) 중복 연산자(+) 발생 ++ 이므로 +로 변경
(Warning) ; 입력 누락. ; 입력 추가
<프로그램에 적힌 코드> target := operand1 + operand2 * 3
ID: 3 ; CONST: 1 ; OP: 2 ;
(OK)
Result ==> 1operand1 : 3 ; operand1 : Unknown ; operand2 : Unknown ; target : Unknown ;
```

-에러 1번(;누락), 에러 2번(잘못된 식별자이름), 에러 7번(++ 사용) 발생

6) test6 파일.

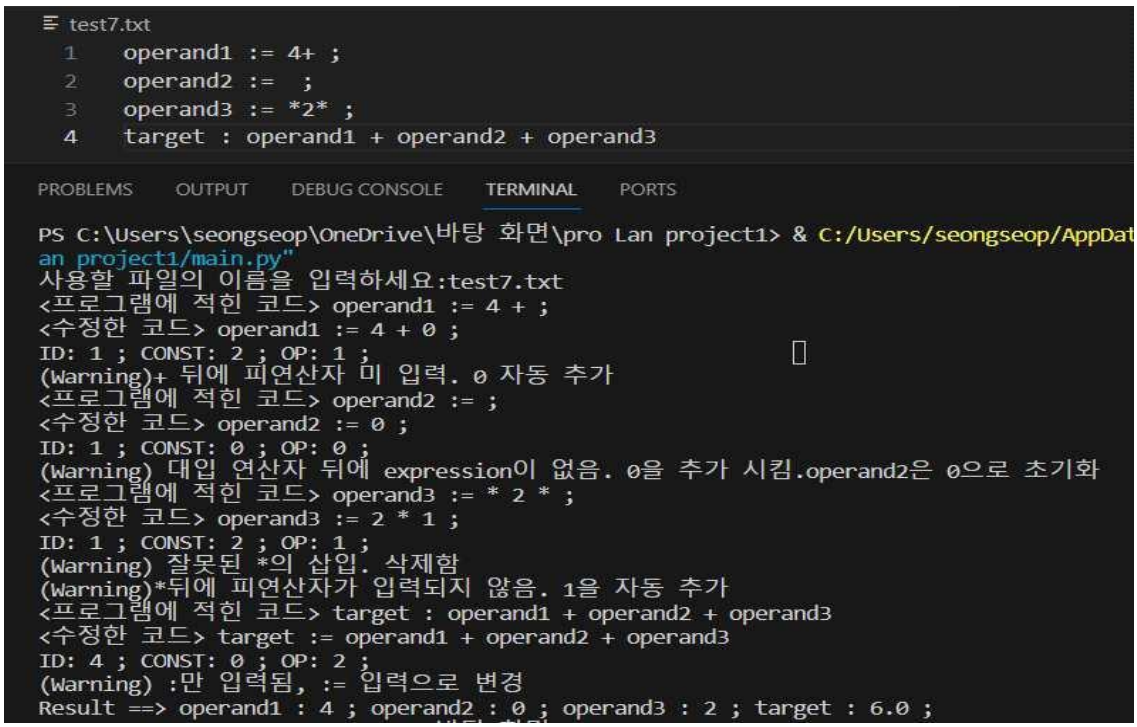
```
test6.txt
1    operand1 := 1;
2    operand2 := (operand1 * 3 + 2 ;
3    target := operand1 + operand2 * 3

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\seongseop\OneDrive\바탕 화면\pro Lan project1> & C:/Users/seongs
 project1/main.py"
사용할 파일의 이름을 입력하세요:test6.txt
<프로그램에 적힌 코드> operand1 := 1 ;
ID: 1 ; CONST: 1 ; OP: 0 ;
(OK)
<프로그램에 적힌 코드> operand2 := ( operand1 * 3 + 2 ;
<수정한 코드> operand2 := ( operand1 * 3 + 2 ) ;
ID: 2 ; CONST: 2 ; OP: 2 ;
(Warning) ) 가 안 닫힘. ) 추가.
<프로그램에 적힌 코드> target := operand1 + operand2 * 3
ID: 3 ; CONST: 1 ; OP: 2 ;
(OK)
Result ==> operand1 : 1 ; operand2 : 5.0 ; target : 16.0 ;
```

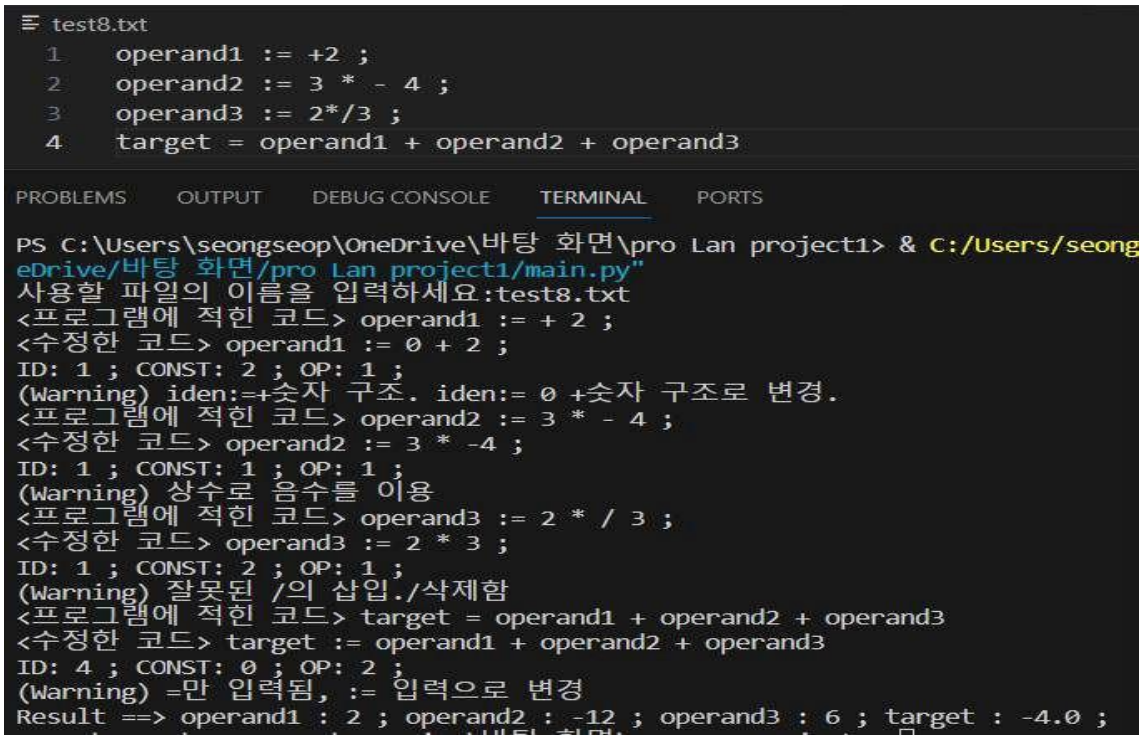
- 에러 14번(')'가 안 닫힘) 발생

(7) test7 파일



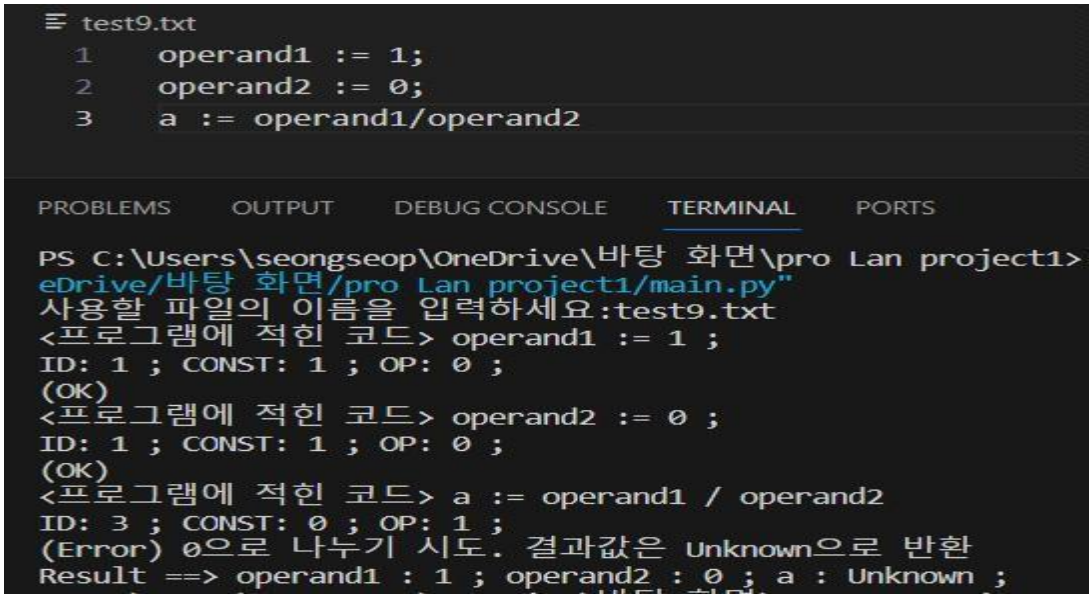
– 에러3번(: 입력), 에러5번(= 뒤에 연산식 미입력), 에러8번(+ 뒤에 피연산자 미입력), 에러9번(잘못된 *), 에러 10번(* 뒤에 피연산자 미입력) 발생

(8) test8 파일



–에러 3번(= 이용) 에러 6번(A:+2 꼴 사용) 에러 12번(*/사용) ,에러 15번(음수 상수 사용) 발생

(9) test9 파일



– 에러 11번(0으로 나눔) 발생

4. 코드 설명

- 더 자세한 설명은 코드 주석 참고

-#3~#20: 프로그램에서 사용하는 주요 전역 변수들 선언
 코드를 가리 키는 위치, 토큰의 개수, 다음 문자, lexeme, charclass, 식별자 개수, 상수 개수, 연산자 개수, 에러 상태, 식별자 사전, 모든 토큰을 모아 놓은 배열

-#21~#28: 상수들 정의
 token_code 들 및, 에약어 종류 정의, LETTER,DIGIT,UNKNOWN,EOF 상수 정의

-#29~#138 lexeme 관련 함수 정의
 getchar(): 문자 하나를 입력 받아서 그 문자의 charclass를 파악한다.
 getNonBlank(): 공백이 들어오면 공백을 무시하고 공백이 아닌 문자가 나올 때까지 계속 읽는다.
 addChar():lexeme에 문자 하나를 추가한다.
 lookup(): unknown 문자 ch가 catogory에 속하는지 확인한다.
 lexical(): 코드를 lexeme으로 분리하는 함수다

-#139~#473 구문 분석 관련 함수 정의
 initialize(): 구문 분석에 필요한 전역 변수를 초기화 하는 역할
 printinfo(): 분석한 문장, 각 요소의 개수, error 상태를 출력하는 함수
 print_indent_value(): 식별자 값 출력
 statements(): statements -> <statement> {; <statement>}를 분석하기 위한 함수
 statement(): <statement> -> <ident> := <expression>를 분석하기 위한 함수
 expr(): <expression> -> <term> {(+|-) <term>}를 분석하기 위한 함수
 term(): <term> -> <factor> {(*|/) <factor>}를 분석하기 위한 함수
 factor(): #<factor> -> (<expression>) | 식별자 | 정수 상수 를 분석하기 위한 함수

-#474~#488 main() 함수
 사용자로 부터 파일명을 입력 받고 파일을 읽기 모드로 열며 코드를 읽는다. statements() 함수를 실행시켜 구문 분석을 시작한다.