# Programming Assignment 1

## 개요

✏️ 교재에 따르면 재귀 하강 파서는 EBNF 와 잘 맞아 떨어지기 때문에, 과제에서 주어진 LL(1)문법을 EBNF로 변환하여 이를 Python으로 구현하였습니다.
또한 각 <요소> 별 간의 각 토큰을 구별하기 위해 과제의 요구조건과 부합하게 입력 스트림에서 ASCII 코드값이 32 이하인 것은 모두 white-space로 간주되며, 각 token을 구별하는 용도 이외에는 모두 무시됩니다.

추가로 보완할 점은, 예외 처리하지 못한 케이스가 존재하여 이와 관련된 부족한 공부를 보충할 예정입니다.

## 실행 방법

- Windows & macOS

```
python3 main.py (텍스트 파일 이름)

ex) python3 main.py input1.txt
```

### 과제 첨부 예시 실행 결과

```
operand1 := 3
ID : 1, CONST : 1, OP : 0
(OK)
Result ==>  {'operand1': 3}

operand2 := operand1 + 2
ID : 2, CONST : 1, OP : 1
(OK)
Result ==>  {'operand1': 3, 'operand2': 5}

target := operand1 + operand2 * 3
ID : 3, CONST : 1, OP : 2
(OK)
Result ==>  {'operand1': 3, 'operand2': 5, 'target': 18}
```

```
operand2 := operand1 + 2
ID : 2, CONST : 1, OP : 1
Error : 정의 되지 않은 변수 operand1가 참조 됨
Result ==>  {'operand1': 'Unknown', 'operand2': 'Unknown'}

target := operand1 + operand2 * 3
ID : 3, CONST : 1, OP : 2
Error : 정의 되지 않은 변수 operand1가 참조 됨
Error : 정의 되지 않은 변수 operand2가 참조 됨
Result ==>  {'operand1': 'Unknown', 'operand2': 'Unknown', 'target': 'Unknown'}
```

```
operand1 := 1
ID : 1, CONST : 1, OP : 0
(OK)
Result ==>  {'operand1': 1}

operand2 := ( operand1 * 3 ) + 2
ID : 2, CONST : 2, OP : 2
(OK)
Result ==>  {'operand1': 1, 'operand2': 5}

target := operand1 + operand2 * 3
ID : 3, CONST : 1, OP : 2
(OK)
Result ==>  {'operand1': 1, 'operand2': 5, 'target': 16}
```

```
operand1 := 3
ID : 1, CONST : 1, OP : 0
(OK)
Result ==>  {'operand1': 3}

operand2 := operand1 + 2
ID : 2, CONST : 1, OP : 1
WARNING : 2개 이상의 중복 연산자를 사용하였습니다
ERROR : <factor>에서 파스 트리의 <Token.ADD_OP>를 제거합니다
ERROR : +를 제거하며 파스 트리를 복구 및 계산합니다
Result ==>  {'operand1': 3, 'operand2': 5}

target := operand1 + operand2 * 3
ID : 3, CONST : 1, OP : 2
(OK)
Result ==>  {'operand1': 3, 'operand2': 5, 'target': 18}
```

# 에러 처리 목록

1. 중복 연산자
   - 연산자가 여러 번 나올 경우 첫번째 연산자를 제외한 나머지 연산자 제거
     - 예시 ) $operand := 1 * / - - - - := 2 \ \Rightarrow \ operand := 1 * 2$주
2. 중복 식별자 및 상수
   - 식별자 <ident> 혹은 상수 <const>가 여러번 나올 경우 첫번째 식별자 / 상수를 제외한 나머지를 제거
     - 예시 ) $op1 := op2op3123 + 23 - 1abc \ \Rightarrow \ op1 := op + 23 - 1$
3. 할당받을 변수 <ident> 또는 할당 연산자 <assign_op>가 없는 경우
   - 할당 받을 변수 <ident>가 없는 경우 ‘ErrorVar’라는 이름의 변수가 값을 할당 받게 됨
     - 예시) $:= 1 + 2 + 3 \ \Rightarrow \ ErrorVar := 1 + 2 + 3$
   - 할당 연산자 <assign_op>가 없는 경우 이를 생성함
     - 예시) $op1 + op2 + op3 \ \Rightarrow ErrorVar := op1 + op2 + op3$
4. 식별자 <ident>의 이름이 C언어의 식별자 규칙과 맞지 않는 경우 Warning 메시지를 출력함
   - 길이가 1 ~ 31자 이며, `_`, `알파벳 소문자, 알파벳 대문자, 숫자` 로 이루어진다
   - 숫자로 변수명을 시작할 수 없다

- C언어 예약어를 변수명에 포함시킬 수 없다

5. LEFT_PAREN 과 정합성이 맞는 RIGHT_PAREN이 없는 경우 RIGHT_PAREN을 <statement>의 맨 뒤에 추가 해준다
   - 예시) $var := (10 + 20 + 30 \quad \Rightarrow. \quad var := (10 + 20 + 30)$
6. Statements의 말미에 SEMI_COLON으로 끝나는 경우, Statement 사이에 SEMI_COLON이 중복으로 존재하는 경우
   - 전자의 경우 이를 제거하고, 중복의 경우 하나를 제외한 모든 SEMI_COLON을 제거 한다.
   - 예시) $var1 := 10 + 20;;; var2 := 1 + 2; \quad \Rightarrow. \quad var1 := 10 + 20; var2 := 1 + 2$

## 예외 처리 하지 못한 케이스

- 정합성이 맞는 RIGHT_PAREN을 제거해주는 경우
  - 예시) $var := 10 + 20 + 30)$