1. **How Syntax analyzer works with lexical analyzer**

* **Lexical Analyzer to Syntax Analyzer**
  + **Lexical Analyzer의 출력 형식을 json으로 수정하여 여러 정보를 넘길 수 있게 했다.**
  + **에러 리포팅을 위해서 토큰에 그 토큰이 끝나는 위치(소스 코드 안에서의)를 지정하였다.**
  + **에러 리포팅을 위해 원본 소스코드를 같이 넣는다.**
* **Syntax Analyzer의 동작원리**
  + **실제 Parse를 진행하는 부분은 SyntaxAnalyzer 클래스가 담당한다.**
  + **Parse에 필요한 Data를 가지는 부분은 SLRTable 클래스가 담당한다.**
  + **프로그램 시작시 Lexical Analyzer에서 만든 json파일을 읽어서, 작업을 진행한다.**
* **실행을 위해 필요한 것**
  + **pandas 모듈 (python)**
    - **SLR TABLE을 csv로 출력하기 위해 필요하다.**
* **SLRTable 클래스**
  + **다음을 Member 변수로 가진다.**
    - **follow\_sets : 심볼에 대한 follow set을 가지는 dictionary다.**
      * **key = 심볼, value = follow set의 list**
    - **transitions : DFA의 transition 정보를 가지는 defaultdict(dict)이다.**
      * **key = 출발 state**
      * **value = dict(key = 트랜지션에 해당되는 심볼, value = 목표 state)**
    - **change\_rules : 모든 Change rule을 리스트로 저장한다.**
      * **Change rule을 사용하는 곳은 이 리스트에서의 index를 가지고 있다 활용할 때 이 리스트를 참조한다.**
    - **non\_terminals : 논터미널들을 리스트로 저장한다.**
    - **terminals = 터미널들을 리스트로 저장한다.**
    - **state\_to\_change\_rules : DFA에 해당되는 change\_rule들의 index와 shift 위치를 리스트로 저장하는 dfa이다.**
      * **key : dfa의 번호**
      * **value = (change\_rule\_index, shift\_index)을 저장하는 list**
    - **action\_table : SLRTABLE의 Action Table을 저장하는 defaultdict(dict)**
    - **goto\_table : SLRTABLE의 GOTO 테이블을 저장하는 defaultdict(dict)**
  + **작성한 DFA의 정보를 넣고, SLRTable에 넣을 수 있는 형식으로 가공한다.**
    - **change rule을 넣고, 그 State에 맞는 change\_rule을 설정한다.**
    - **Trasnsition Rule을 넣어준다.**
    - **Start State는 “S”로 고정한다. (HARD CODING)**
  + **FollowSet, Terminal, Nonterminal을 넣어준다**
  + **위 정보를 이용해서 GoTo, Action 테이블을 Build한다.**
    - **Action 테이블은 (액션 타입, 액션 VALUE)의 튜플을 넣어준다.**
    - **수업시간에 설명해주신 Rule을 충실히 따라, 코드로 생성한다.**
  + **빌드한 테이블을 확인하기 위해 csv로 export 한다.**
  + **Syntax Analyzer 인스턴스를 이때까지 생성한 SLR TABLE을 이용해 생성한다.**
  + **Syntax Analyzer의 Parsing을 진행한다.**
* **Syntax Analyzer**
  + **다음을 Member 변수로 가진다**
    - **slr\_table : 위에서 만든 slr\_table이다.**
    - **state\_stack : 현재 탐색중인 state의 stack이다 (list로 구현) [0]으로 초기화하는데, 이것은 initial state이다.**
    - **passed\_state, passed\_change\_rule**
      * **이 검증 데이터셋은 어느정도 테이블을 사용하고 있는지 검증용으로 사용. 지나온 state와 change\_rule를 저장.**
    - **input\_symbols : 현재의 symbols(Non/terminal)들을 저장한다.**
      * **처음에는 토큰들로 초기화**
      * **이것이 Start State가 되면 종료.**
      * **NonTerminal은 그 Nontermal이 만들어진 가장 뒤의 Symbol이 그 Nonterminal의 원래 코드에서의 위치가 된다.**
    - **shifter\_index : 현재 보고있는 input\_symbols에서 어느 위치까지 shift해서 보고 있는지를 저장한다.**
  + **parse\_one 함수를 한번 호출할때마다 한번씩 parsing을 진행한다.**
  + **sfift\_index를 보고 현재 검사해야할 symbol을 가져온다.**
  + **non\_terminal이면 GOTO, terminal이면 ACTION을 실행한다.**
  + **해당되는 것이 없으면 ERROR를 발생한다.**
  + **parse\_one은** 
    - **True :파싱이 정상적으로 진행중이다.**
    - **“END” : 파싱이 완료되었고, 성공적이다.**
    - **(False, symbol): 파싱 실패, 실패한 위치에서의 symbol을 반환**
    - **이 Symbol은 에러 리포트를 작성할 때 쓰인다.**
  + **parse\_one의 과정 세부는 수업시간의 rule을 따른다.**
    - **변환 규칙에서 사용한 Symbol 수 만큼 Pop한다**
    - **SHIFT 액션은 SHIFT한 후 State를 이동한다. 등등...**
* **OVERALL PROCEDURES**
  + 입력받은 파일 이름에 해당되는 파일이 있는지 확인.
  + **사전 정의된 값으로 SLRTable 클래스 생성후 초기화**
  + **SLRTable의 action, goto 테이블을 csv로 export한다.**
  + **SyntaxAnalyzer 클래스를 위 SLRTable로 생성**
  + **Parse 진행 후 결과 출력.**
    - **성공이면 메시지 출력후 종료**
    - **실패이면 에러 리포트 작성 후 종료**