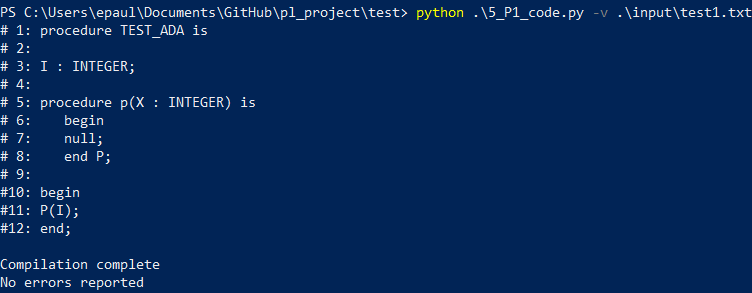
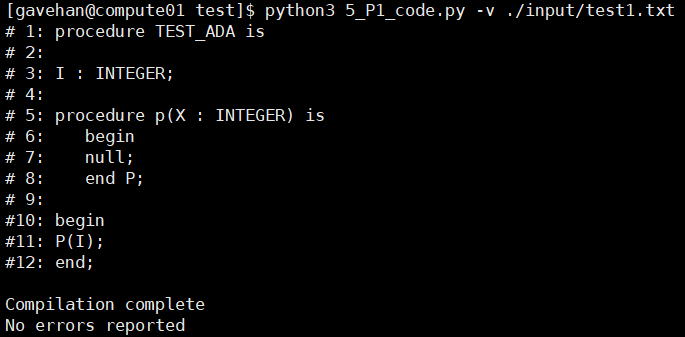
|  |  |
| --- | --- |
| 텀프로젝트 P1 설명서  CSI3103-01 프로그래밍언어구조론 | Python을 이용한 Syntax Analyzer 구현  5조: 김재민, 김준한, 마준영  2020년 10월 14일 |

텀프로젝트 P1 설명서

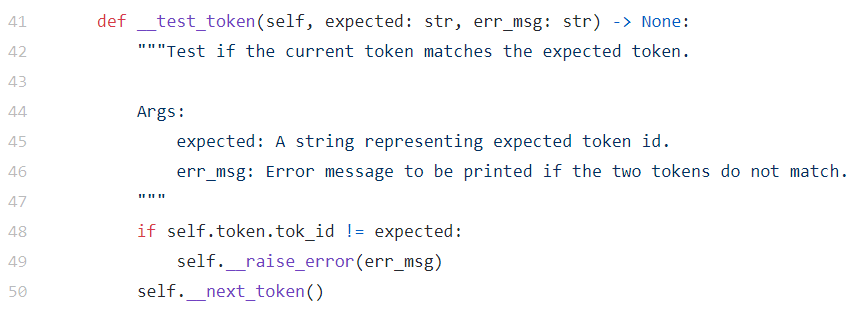
# 사용법

1. Windows
   1. Windows PowerShell를 실행한다.
   2. Windows PowerShell에서 cd {프로젝트 소스코드 경로} 명령어를 실행한다.
   3. Windows PowerShell에서 python ./5\_P1\_code.py {테스트 파일 경로} 명령어를 실행한다.  
      (경우에 따라, python이 아니라 python3일 수도 있다.)  
      
   4. 자세한 CUI 사용방법은 현재 문서 “모듈 소개”에서 확인할 수 있다.
2. Unix 계열
   1. cd {프로젝트 소스코드 경로} 명령어를 실행한다.
   2. python ./5\_P1\_code.py {테스트 파일 경로} 명령어를 실행한다.  
      (경우에 따라, python이 아니라 python3일 수도 있다.)  
      
   3. 자세한 CUI 사용방법은 현재 문서 “모듈 소개”에서 확인할 수 있다.

# EBNF 구현

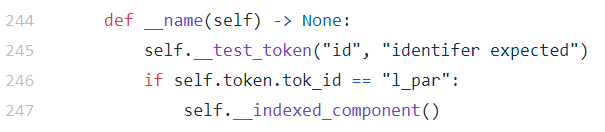
1. Recursive descent parser 구현전략
   1. 리터럴 항목

Parser 모듈의 \_\_test\_token 메서드를 통해 확인한다. 해당 메소드는 현재 parsing 중인 토큰과 현재 문법에서 기대하고 있는 토큰을 비교를 한다. 두 토큰이 일치하지 않으면, 예외를 호출하고 오류 메시지를 출력한다.



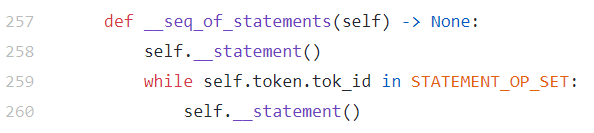
* 1. [ ] 문법을 갖는 항목

대괄호 안에 있는 phrase 또는 phrase들의 첫번째 토큰과 현재 토큰을 비교해본다. 만약, 현재 토큰이 대괄호 안 phrase의 첫번째 토큰 또는 토큰들 중 하나와 일치한다면, 해당 대괄호 안 phrase로 descent한다. “name” phrase의 구현을 해당 전략의 예시로 제공한다.



* 1. { } 문법을 갖는 항목

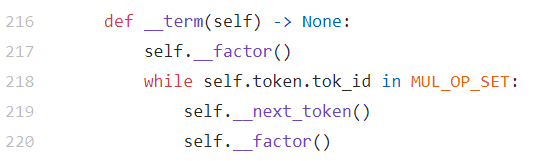
중괄호 안에 있는 phrase 또는 phrase들의 첫번째 토큰과 현재 토큰을 비교해본다. 만약, 현재 토큰이 중괄호 안 phrase의 첫번째 토큰 또는 토큰들 중 하나와 일치한다면, 해당 중괄호 안 phrase에 대해서 while문을 통하여 반복적으로 확인한다. “sequenceOfStatements” phrase의 구현을 해당 전략의 예시로 제공한다.



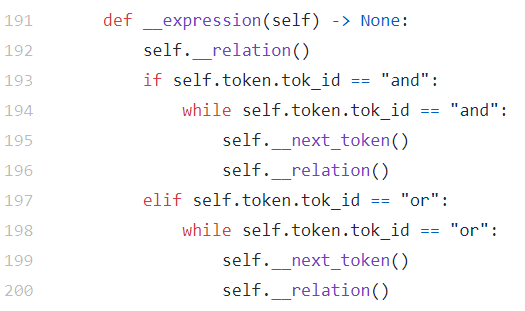
* 1. | 문법을 갖는 항목

해당 문법을 접근한 방식은 두 가지이다.

연산자와 같은 terminal 또는 “statement” phrase의 경우에는 해당 토큰들로 파이썬 set를 구성하고, 현재 토큰의 set에 대한 멤버십을 확인하는 접근 방식이 있다. “multiplyingOperator” phrase와 “term” phrase 의 구현을 해당 전략의 예시로 제공한다.

또 다른 전략은 조건문을 통해 선택지 phrase 중 어느 phrase가 토큰과 일치하는지 확인하는 방식이 있다. 이 전략을 이용한 phrase들은 nonterminal이므로, 리터럴를 통해 선택지 구분이 어려운 경우에는 2번과 3번 전략과 유사하게, 현재 토큰이 선택지 phrase의 첫번째 토큰 또는 토큰들 중 하나와 일치하는지를 판별한다. “expression” phrase의 구현을 해당 전략의 예시로 제공한다.



1. EBNF 문법의 구현방식
   1. identifier

Ada 언어에서 변수 이름은 알파벳으로 시작하고, 알파벳, 숫자, “\_”로만 구성되며, reserved keyword가 아닌 단어로 정의한다.

현재 프로젝트에서 identifier는 첫 문자가 대소문자 구별없이 알파벳이고, 그 뒤 문자는 숫자, 대소문자 구별 없이 알파벳, “\_” 중 하나로 연속적으로 구성되며, 해당 단어가 reserved keyword가 아닌 단어로 정의한다.

* 1. numericLiteral

Ada 언어에서 정수, 분수, 실수, 복소수 자료형을 지원한다.

현재 프로젝트에서 numericLiteral는 정수 자료형으로 제한한다. 따라서, numericLiteral는 첫 문자가 숫자이고, 그 뒤 문자도 모두 숫자로 연속적으로 구성된 단어로 정의한다.

* 1. Recursive descent parser 구현전략 사용.  
     4번 항목부터는 recursive descent parser 전략 외 방식을 이용한 구현대상에 대한 설명이다.
  2. compilation

eof 토큰에 대해 test\_token 메서드 호출한다. (프로그램의 정상 종료를 위해 추가)

chario.report\_errors 메서드 호출한다. (프로그램의 정상 종료와 오류의 수를 출력)

* 1. objectDeclaration, numberDeclaration

두 phrase 모두 identifierList로 시작한 뒤 “:” 토큰을 요구하고, “;” 토큰으로 종료한다는 공통점에  
착안하여 하나로 통합했다. (obj\_num\_declaration)

* 1. statement, simpleStatement, compoundStatement

두 phrase의 구현 관점에서 구분이 불가하다는 점에 착안하여 상위 phrase인 statement로 통합했다.

* 1. assignmentStatement, procedureCallStatement

두 phrase 모두 name으로 시작하고, “;” 토큰으로 종료한다는 공통점에 착안하여 하나로  
통합했다. (assign\_call\_statement)

procedureCallStatement에서 actualParameterPart는 indexedComponent와 문법적으로 동일한 phrase이므로, 두 phrase 중 indexedComponent만 구현하여 사용했다.

* 1. loopStatement, iterationScheme

iterationScheme은 loopStatement에서만 사용하므로, 상위 phrase인 loopStatement로 통합했다.

# 모듈 소개

1. CUI (5\_P1\_code.py)
   1. Logger 클래스

stdout(터미널 화면)과 주어진 파일 경로의 로그 파일에 동시 출력 기능을 제공한다.

* 1. receive\_args 함수

커맨드라인에서 옵션을 받는다. 옵션은 “-v”, “-o”가 있고, 필수 입력 항목은 TinyAda 소스 프로그램 파일 경로이다. “-v” 옵션을 준 경우, 입력 소스 프로그램 전체가 출력되며, 필수로 오류 보고를 출력한다. “-o” 옵션을 준 경우, syntax analyzer의 출력을 stdout(터미널 화면)과 주어진 파일 경로의 로그 파일에 동시에 출력한다.

* 1. main

커맨드라인에서 받은 옵션들과 입력 소스 프로그램 파일로 Chario, Scanner, Parser 클래스들의 인스턴스를 각각 생성한다. 그 후에, Parser 클래스에서 parsing을 시작한다. Parsing 중, syntax 오류가 발생한 경우, 예외 처리를 통해서, 오류를 출력하고 종료한다. Parsing 중 오류가 syntax 오류가 발생하지 않은 경우에, lexical 오류가 있다면 보고하고, 만약 오류가 없었다면 출력이 없다.

1. Token (token.py) – Token 클래스를 통해 parsing 중 여러 모듈간 상호작용을 용이하게 함.
   1. Token 클래스

개념적으로 토큰을 상징한다. 속성으로 lit, tok\_id를 갖는다.

lit은 문자열 자료형으로 코드를 저장하고, tok\_id는 문자열 자료형으로 토큰 종류를 저장한다. tok\_id를 통해 표현 가능한 토큰 종류는 reserved keyword, operator이거나 정수 자료형, identifier, type, eol, eof 중 하나이다.

* 1. lit\_to\_tok 함수

문자열 자료형의 코드를 Token 클래스 인스턴스로 변환한다.

1. Chario (chario.py)
   1. Chario 클래스

소스 프로그램을 읽으면서, 계속 문자를 Scanner에게 전달하는 역할을 한다. 속성으로 src, is\_verbose, line, err\_count, column, line\_count를 갖는다.

src는 문자열을 요소로 갖는 리스트로 소스 프로그램을 저장한다. is\_verbose는 불 자료형으로 더 자세한 출력 여부를 저장한다. line, column, line\_count 속성을 통해서 소스 프로그램에서 현재 몇 번째 줄에서 몇 번째 문자를 읽고 있는지 나타낸다. err\_count의 경우, parsing 중 발생한 오류의 수를 저장한다.

클래스 init 도중에 소스 프로그램을 열지 못하는 오류는 해당 클래스에서 처리한다. 또한, 해당 클래스는 필요시 오류 메시지를 출력하는 역할도 하며, 자세한 출력일 경우, 소스 프로그램도 출력하는 역할도 한다.

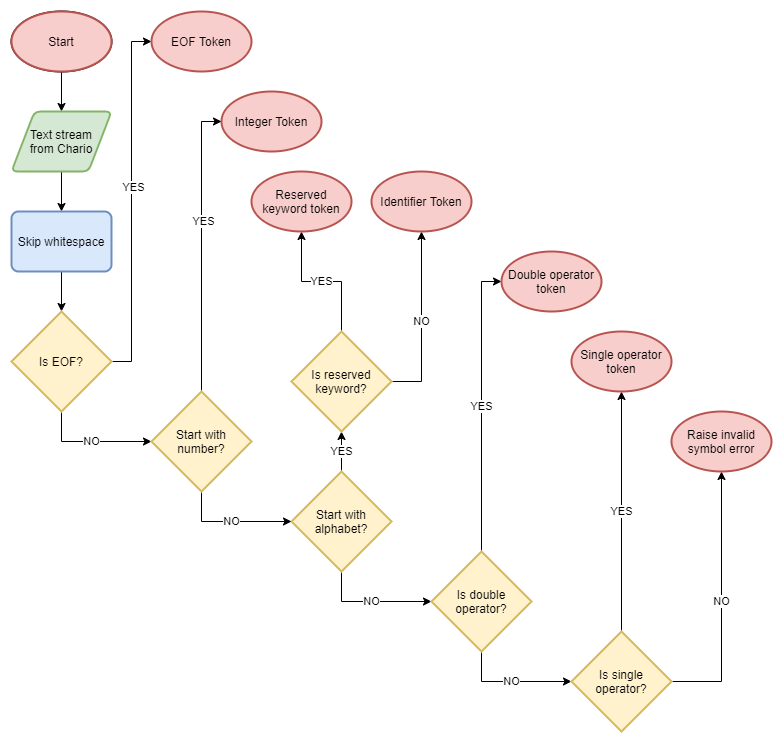
1. Scanner (scanner.py)
   1. Scanner 클래스

Chario에서 전달받은 문자 스트림을 토큰으로 변환하는 역할을 한다. 속성으로 chario, buffer, char를 갖는다.

chario 속성은 주어진 소스 프로그램으로 생성된 Chario 클래스의 인스턴스이다. buffer는 문자를 요소로 갖는 리스트로, 토큰 변환하기 전 리터럴 코드를 저장한다. char는 Chario 클래스 인스턴스에서 전달 받은 문자이다.

토큰을 추출하는 과정 중 lexical 오류를 발견할 경우. 오류 메시지를 Chario에 전달하고, 다음 토큰을 추출하여 한다.

TinyAda의 예약어와 연산자 길이와 구성 문자 종류가 상이하므로, 해당 전략으로 토큰을 추출한다.



1. Parser (parser6.py)
   1. Parser 클래스

개념적으로 TinyAda의 EBNF 문법을 바탕으로, recursive descent 전략을 통해 syntax error를 발견하는 parser이다. Scanner로부터 토큰 스트림을 받고, 오류는 Chario에게 전달한다. 속성으로 chario, scanner, token을 갖는다.

chario 속성은 주어진 소스 프로그램으로 생성된 Chario 클래스의 인스턴스이다. Scanner 속성은 해당 chario을 바탕으로 하는 Scanner 클래스의 인스턴스이다. token 속성은 현재 parsing 대상인 토큰을 저장한다.

Parsing 중 syntax 오류를 발견할 경우. 오류 메시지를 Chario에 전달하고, parsing을 중지하고 종료한다.

Parser의 구체적인 원리와 전략은 “EBNF 구현”에서 확인할 수 있다.

1. 프로젝트의 파이썬 모듈에 있는 주석을 통하여 추가적인 설명, 사용 예시에 대해 확인할 수 있다.

# 후기

**김재민 (담당 파트: Chario, Parser 중 statement)**

이번 프로젝트를 통해서 프로그래밍 언어의 syntax 구조를 construct하는 방식과, 언어가 text를 파싱하는 과정을 직접 작성해보며 이해할 수 있었다. 또한 단체 프로젝트를 통해서 서로가 협동하며, 도와주며 원하는 결과를 만들어내는 과정이 매우 만족스러웠다.

**김준한 (담당 파트: Token, CUI, Parser 중 expression)**

이번 프로젝트를 통해서, 여러 회의를 통해서, 프로젝트의 틀과 기반을 (git, 개발환경, linter, 스타일가이드) 함께 고민하면서 만드는 과정이 쉽지는 않았지만, 유익한 경험을 했다. 또한, 프로그래밍 언어가 정상적으로 작동하기 위해 syntax 관련한 여러 요소들이 유기적으로 작동하는 것을 직접 해볼 수 있어서 좋았다.

**마준영 (담당 파트: Scanner, Parser 중 declaration)**

이번 프로젝트로 인해 프로그래밍 언어의 syntax에 대해 이해할 수 있었을 뿐만 아니라, 공동 개발을 위한 git이나 스타일 가이드에 대해서도 알 수 있었다. 프로젝트를 통해 이와 같은 새로운 지식들을 얻을 수 있어서 좋았다.