# 컴파일러 제작

컴파일러 제작의 이해를 돕기 위해 작성한 문서.

## 참고 자료

<http://www.kwangsiklee.com/2018/06/%ec%bb%b4%ed%8c%8c%ec%9d%bc%eb%9f%ac-%ea%b0%95%ec%9d%98-1-%ea%b0%9c%ec%9a%94/>

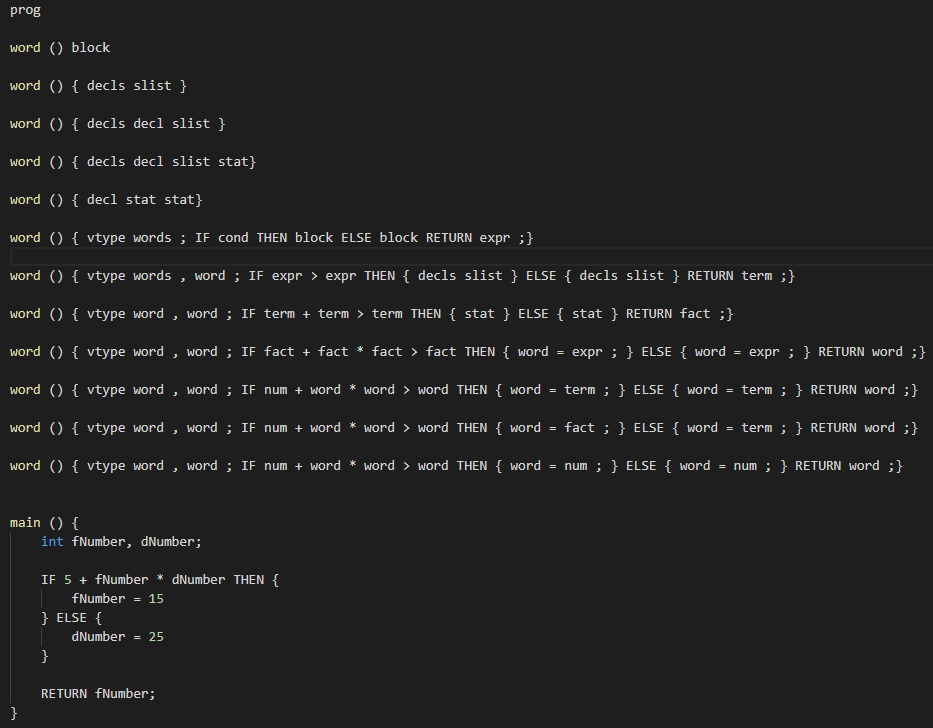
<https://crystalcube.co.kr/107?category=461606>

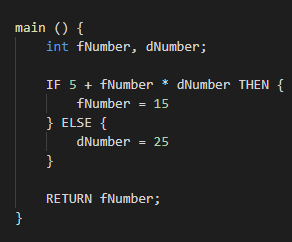
# PDF 이해

PDF의 표 1 GRAMMAR은 BNF 표기법에 따라 작성된 문법

왼쪽의 prog, decls, … 등은 토큰(token)이다.

이 외에 “{“, “(“, “;” 등에도 토큰 이름을 부여하여야 한다.



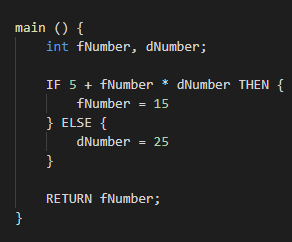


이런 코드가 있다고 가정하자.

# 단계

* Source Program
* Lexical Analyzer
* Syntax Analyzer
* Semantic Analyzer
* Intermediate Representation
* Code Optimizer
* Code generator
* Target Program

## Source Program



## Lexical Analysis

* 전처리
* 토큰 인식
* 기호표 구성
* 에러 처리에 대한 진단

### 전처리

스페이스(띄어쓰기), 줄바꿈 등을 모두 제거한다. (우리가 사용하는 문법은 스페이스에 대한 의미가 없다.)

다음은 전처리가 끝난 코드의 모습



### 토큰 인식



정규표현식을 사용하여 토큰 추출을 한다. 다음은 토큰 추출의 결과이다

word : main

구분자 : (

구분자 : )

block 구분자 : {

vtype : int

word : fNumber

구분자 : ,

word : dNumber

구분자 : ;

예약어 : IF

num : 5

연산자 : +

…

구분자 : ;

블록 구분자 : }

이런 형태가 된다.

### 기호표 구성

토큰끼리 의미 구분을 위해 토큰 번호와 토큰 값의 순서쌍으로 표현된 표이다.

(아직 수업에서 언급한 적이 없어서 보류)

## Syntax Analysis

### Parse Tree 그리기

