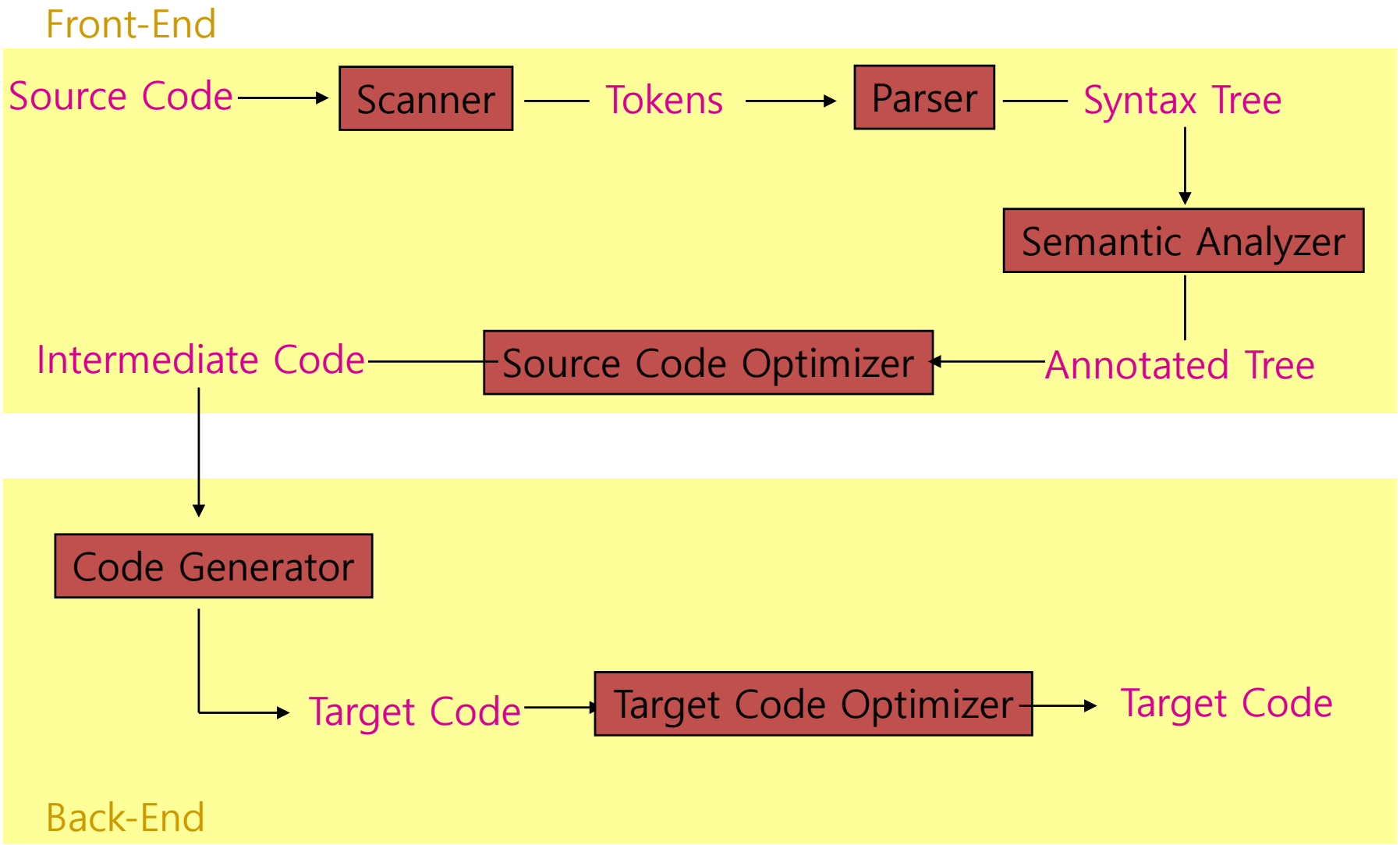
컴파일러 구조

경성대학교

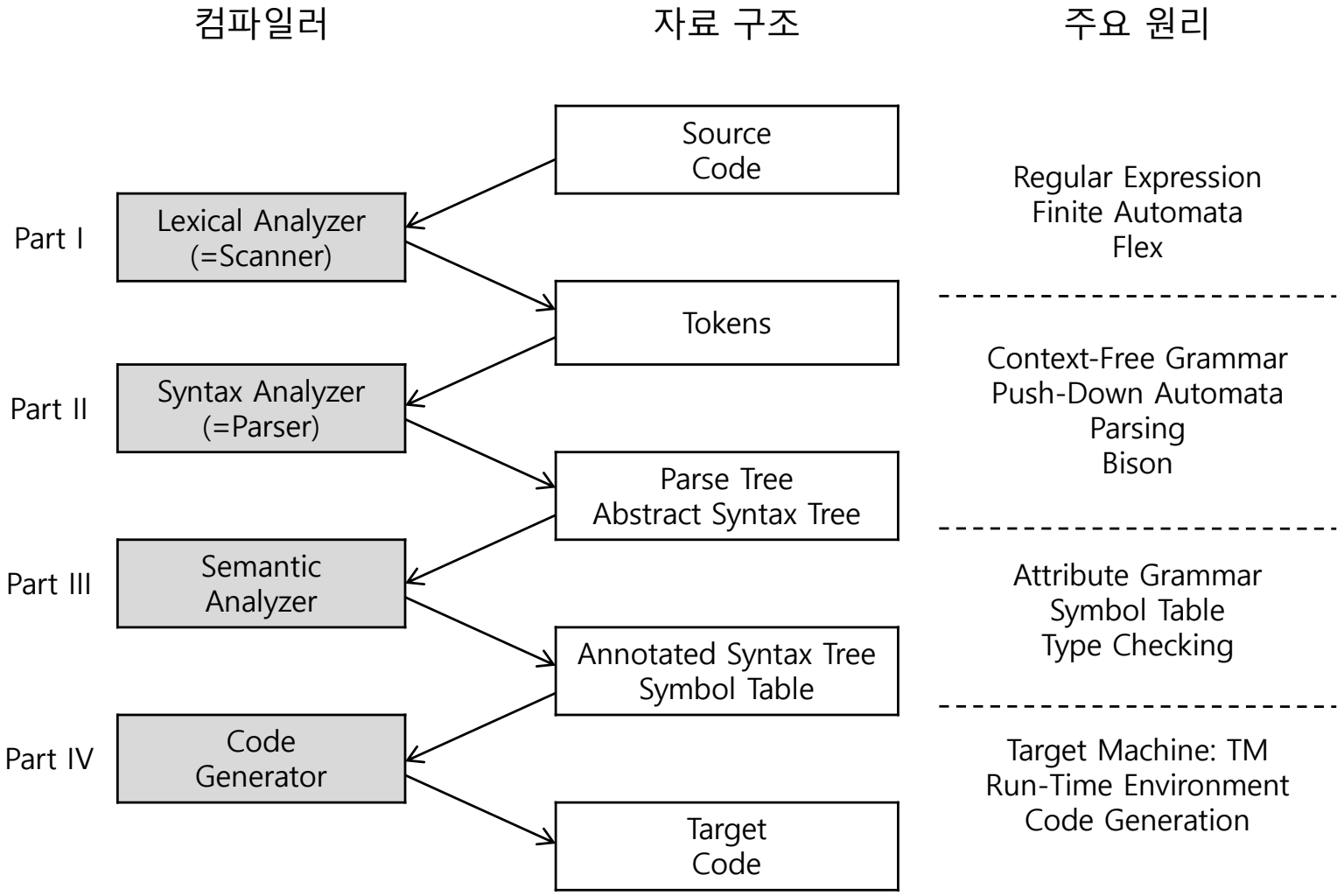
2019년 가을학기

변 석 우
(swbyun@ks.ac.kr)

The Compilation Process



컴파일러 모습


컴파일러
자료 구조
주요 원리
Source Code
Part I
Lexical Analyzer (=Scanner)
Regular Expression
Finite Automata
Flex
Tokens
Part II
Syntax Analyzer (=Parser)
Context-Free Grammar
Push-Down Automata
Parsing
Bison
Parse Tree
Abstract Syntax Tree
Part III
Semantic Analyzer
Attribute Grammar
Symbol Table
Type Checking
Annotated Syntax Tree
Symbol Table
Part IV
Code Generator
Target Machine: TM
Run-Time Environment
Code Generation
Target Code

어휘 분석기(Lexical Analyzer)

- 입력: 소스 코드(source code)

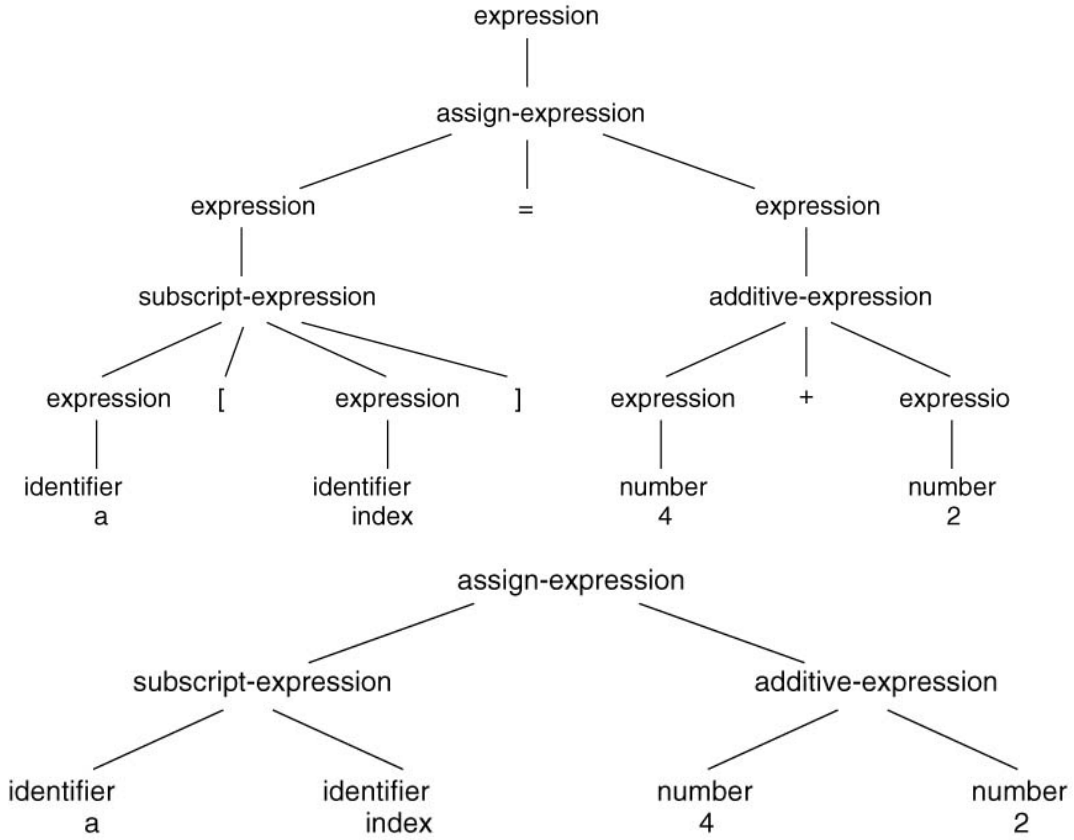
```
a[index] = 4 + 2;
```

- 출력: 토큰(token)

```
token=ID, lexeme="a"
token=LBRACKET, lexeme="["
token=ID, lexeme="index"
token=RBRACKET, lexeme="]"
token=ASSIGN, lexeme="="
token=NUM, lexeme="4"
token=PLUS, lexeme="+"
token=NUM, lexeme="2"
token=SEMICOLON, lexeme=";"
```

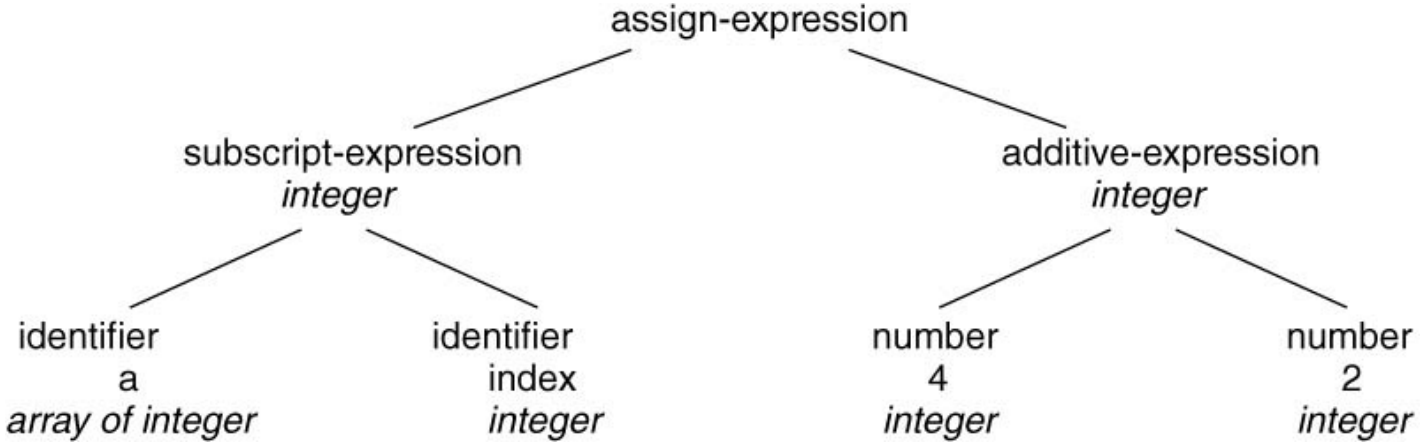
구문 분석기(Syntax Analyzer)

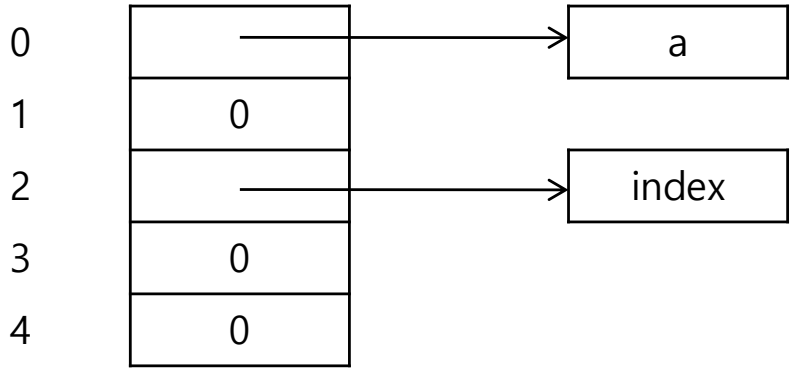
- 입력: 토큰들(tokens)의 스트림(stream)
- 출력: 파스 트리(parse tree), 추상 구문(abstract syntax) 트리



의미 분석기(Semantic Analyzer)

- 입력: 추상 구문 트리
- 출력: 주석(annotated) 구문 트리, 심볼 테이블(symbol table)





코드 생성기(Code Generator)

- 입력: 주석(annotated) 구문 트리, 심볼 테이블(symbol table)
- 출력: 타겟 코드(target code)

```
12: ld 0, -12(fp)       // r0 = index
13: ldc 1, 4            // r1 = 4
14: ldc 2, 2            // r2 = 2
15: add 3, 1, 2         // r3 = r1 + r2
16: add 1, fp, 0        // r1 = &a[index]
17: st 3, -11(1)        // *r1 = r3
```