KNU 4471.043 컴파일러 설계

고상기

4주차

2022 Spring

강원대학교 컴퓨터공학과

3주차 요약

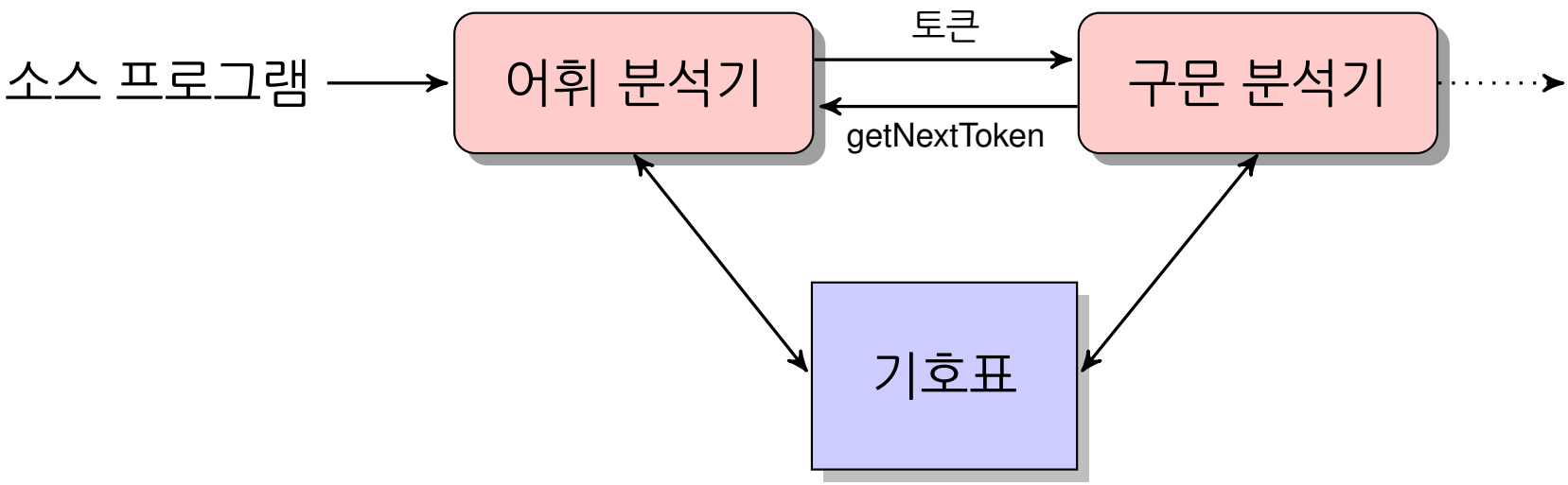
- 형식 언어란?
- 형식 문법이란?
- 정규 표현식
- 유한 오토마타
- 정규 문법과 정규 표현식 그리고 오토마타

4주차 개요

- 어휘 분석lexical analysis
- 토큰token 인식하기
- 어휘 분석기 설계하기
- 어휘 분석기 구현하기

어휘 분석lexical analysis이란?

- 소스 프로그램을 읽어 토큰token이라는 문법적으로 의미있는 최소 단위로 분리하는 작업
- 어휘 분석 도구를 어휘 분석기lexical analyzer 또는 스캐너scanner라고 함
- 어휘 분석기의 기능
  - 소스 코드로부터 토큰들을 순차적으로 인식
  - 오류 메시지와 소스 프로그램을 연관
  - 전처리 수행: 주석comment 제거, 공백 문자 처리, 매크로 처리
  - 기호표symbol table 구성



어휘 분석 관련 용어 정리

- 토큰token: 문법에 존재하는 터미널 기호들로 구성된 문법적으로 의미 있는 최소 단위
- 패턴pattern: 토큰을 서술하는 규칙
- 어휘항목lexeme: 패턴에 의해 매칭된 문자열

토큰의 종류

- 키워드keyword, reserved word: `for`, `if`, `int` 등의 언어에 미리 정의된 단어
- 연산자 기호operator symbol: `+`, `-`, `*`, `/`, `<`, `=` 등의 연산 기호
- 구분자delimiter: `;`, `(`, `)`, `[`, `]` 등 단어와 단어 또는 문장과 문장을 구분하는 기호
- 식별자identifier: `abc`, `num`, `sum`, `i` 등 프로그래머가 정의하는 변수의 이름
- 상수constant: `526`, `2.0`, `0.12422e-12`, `"kangwon"` 등 다양한 타입의 상수

토큰의 패턴pattern

- 토큰을 서술하는 규칙인 패턴은 정규 표현식에 의해 기술된다.
  - C언어의 식별자 패턴: 첫 번째 기호가 영문자 소문자나 대문자 또는 밑줄underscore로 시작하고 두 번째 기호부터는 영문자 소문자나 대문자, 숫자, 밑줄이 올 수 있음

$$
\begin{aligned}
\langle \text{ident} \rangle &::= (\langle \text{letter} \rangle \mid \_)\{\langle \text{letter} \rangle \mid \langle \text{digit} \rangle \mid \_\} \\
\langle \text{letter} \rangle &::= a \mid b \mid \cdots \mid z \\
\langle \text{digit} \rangle &::= 0 \mid 1 \mid \cdots \mid 9
\end{aligned}
$$

- 아래의 문장을 토큰들로 분리해보자.

```
1 ni = ba * po - 60 + ni / (abc + 50);
```

토큰, 어휘항목, 패턴 구하기

- 아래 프로그램 문장에서 토큰, 어휘항목, 패턴을 각각 기술해보자.

```
1 ni = ba * po - 60 + ni / (abc + 50);
```

토큰	어휘항목	패턴
식별자	ni, ba, po, abc	첫 글자가 영문자나 밑줄이고 두 번째부터는 영문자나 숫자 또는 밑줄
연산자	=, +, −, *, /	왼쪽과 동일
상수	60, 50	0부터 9까지의 숫자들로만 이루어진 문자열
구분자	(,), ;	왼쪽과 동일

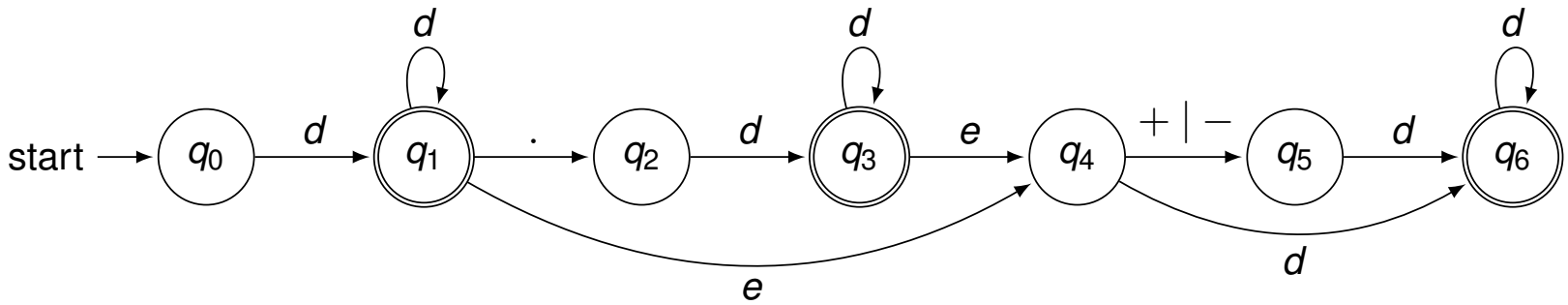
토큰의 인식

- 토큰의 패턴을 기술하는 정규 표현식으로부터 FA를 구성하고 FA를 통해 토큰을 인식한다.

- 예) 부호가 없는 실수 상수의 패턴을 기술하는 정규 표현식

$$d^+ + d^+.d^+ + d^+.d^+e(\varepsilon + \text{`+'} + \text{`}-\text{'})d^+ + d^+e(\varepsilon + \text{`+'} + \text{`}-\text{'})d^+$$

- 위의 정규 표현식을 FA로 변환한 결과



주석 처리하기

- 대부분의 언어는 작성된 코드를 설명하기 위한 주석comment을 제공한다.
- /*와 */ 사이에 작성된 문장을 주석이라고 하자.
- 기호 a는 $*$를 제외한 모든 문자이고 기호 b는 $*$와 $/$를 제외한 모든 문자를 나타낸다고 하자.
- 주석에 대한 패턴은 아래의 정규 표현식을 통해 기술할 수 있다.

$$/*(a+*^{+}b)^{*}(*^{+}/)$$

- 위의 정규 표현식을 FA로 나타내보자.

토큰의 표현

- 효율적인 구문 분석을 위해 토큰을 토큰 번호token number와 토큰 값token value의 순서쌍으로 표현

토큰 번호

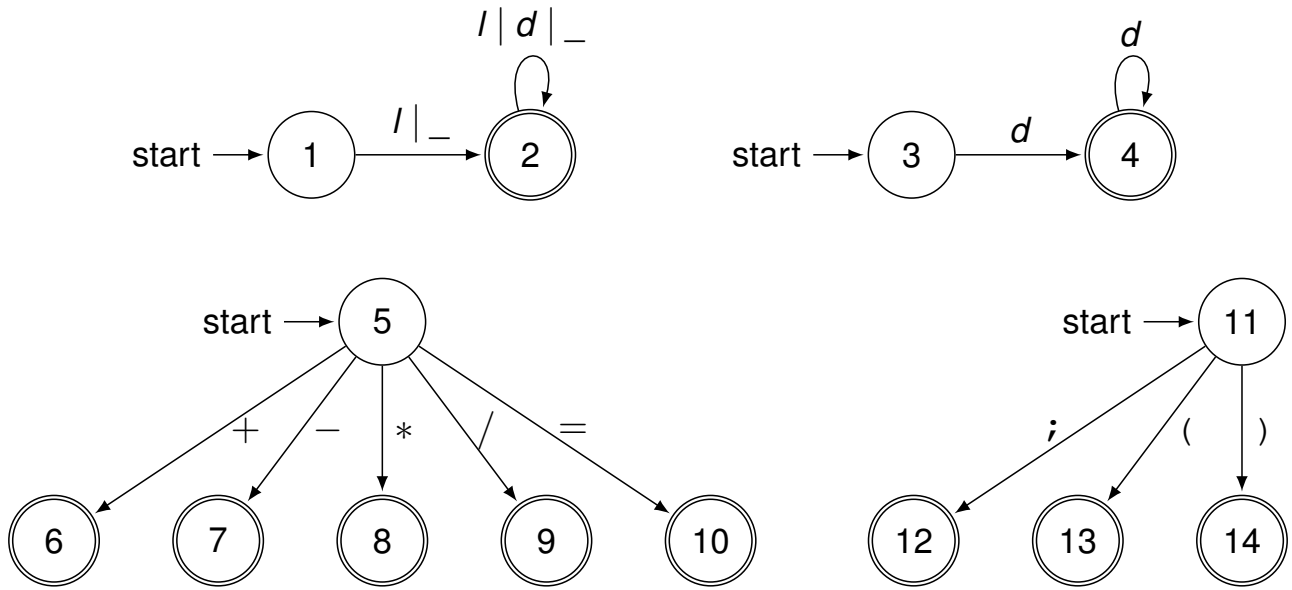
- 각각의 토큰들을 구분하기 위해 고유의 정수 값을 부여한 것

토큰 값

- 토큰들 중 식별자는 하나의 이름으로 여러 번 프로그램에서 사용될 수 있으므로 사용될 때마다 다르게 표현하지 않기 위해 토큰 값을 부여
- 식별자의 경우 기호표에 저장된 주소값, 상수의 경우 실제 값을 부여

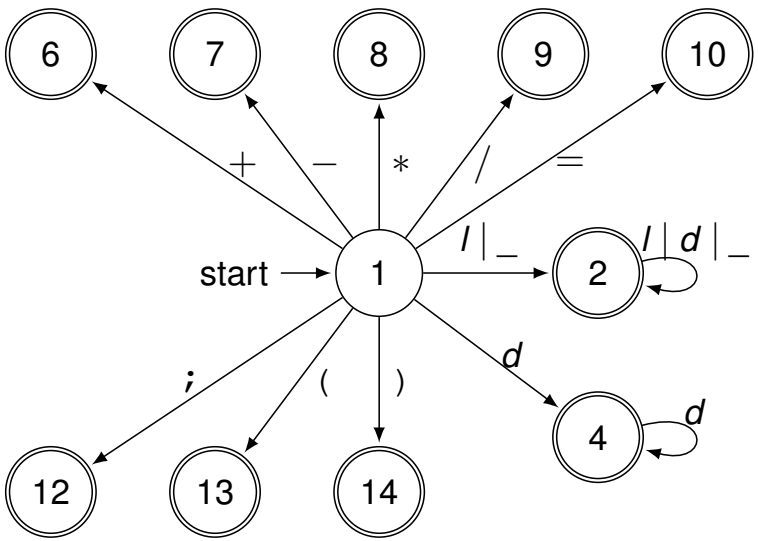
토큰 표현의 예

- 아래는 각각 식별자, 상수, 연산자, 구분자를 인식하는 FA이다.
- 예시 문장 `ni = ba * po - 60 + ni / (abc + 50);`를 아래의 FA를 통해 어휘 분석하여 얻어진 토큰 표현은 다음과 같다.
  - (2,1), (10,∼), (2,2), (8,∼), (2,3), (7,∼), (4,60), (6,∼), (2,1), (9,∼), (13,∼), (2,4), (6,∼), (4,50), (14,∼), (12,∼)



토큰 표현의 예

- 아래는 각각 식별자, 상수, 연산자, 구분자를 인식하는 FA이다.
- 예시 문장 `ni = ba * po - 60 + ni / (abc + 50);`를 아래의 FA를 통해 어휘 분석하여 얻어진 토큰 표현은 다음과 같다.
  - (2,1), (10,∼), (2,2), (8,∼), (2,3), (7,∼), (4,60), (6,∼), (2,1), (9,∼), (13,∼), (2,4), (6,∼), (4,50), (14,∼), (12,∼)



어휘 분석기 구현 방법

- 어휘 분석기를 구현하는 방법
  1. 직접 앞의 단계들을 설계하고 프로그래밍을 통해 구현
  2. Lex, Flex, JavaCC 등의 같은 어휘 분석기 생성기lexical analyzer generator를 통해 구현

- 어휘 분석기 구현 단계
  1. 프로그래밍 언어 문법으로부터 사용되는 토큰과 패턴을 찾는다.
  2. 토큰과 패턴을 가지고 토큰표token table를 만든다.
  3. 각 토큰에 대한 패턴을 정규 표현식으로 나타낸다.
  4. 정규 표현식을 NFA로 변환한다.
  5. NFA를 DFA로 변환한다.
  6. DFA를 최소화한다.
  7. 최소화된 DFA를 프로그래밍 언어를 통해 구현한다.

토큰과 패턴 찾기

$$
\begin{aligned}
\langle \text{Sub C} \rangle &::= \langle \text{assign-st} \rangle \\
\langle \text{assign-st} \rangle &::= \langle \text{lhs} \rangle = \langle \text{exp} \rangle; \\
\langle \text{lhs} \rangle &::= \langle \text{variable} \rangle \\
\langle \text{exp} \rangle &::= \langle \text{exp} \rangle + \langle \text{term} \rangle \mid \langle \text{exp} \rangle - \langle \text{term} \rangle \mid \langle \text{term} \rangle \\
\langle \text{term} \rangle &::= \langle \text{term} \rangle * \langle \text{factor} \rangle \mid \langle \text{term} \rangle / \langle \text{factor} \rangle \mid \langle \text{factor} \rangle \\
\langle \text{factor} \rangle &::= \langle \text{variable} \rangle \mid \langle \text{number} \rangle \mid \text{`(' } \langle \text{exp} \rangle \text{ `)'} \\
\langle \text{variable} \rangle &::= \langle \text{ident} \rangle \\
\langle \text{ident} \rangle &::= (\langle \text{letter} \rangle \mid \_)\{\langle \text{letter} \rangle \mid \langle \text{digit} \rangle \mid \_ \} \\
\langle \text{number} \rangle &::= \langle \text{digit} \rangle \{ \langle \text{digit} \rangle \} \\
\langle \text{letter} \rangle &::= a \mid b \mid \cdots \mid z \\
\langle \text{digit} \rangle &::= 0 \mid 1 \mid \cdots \mid 9
\end{aligned}
$$

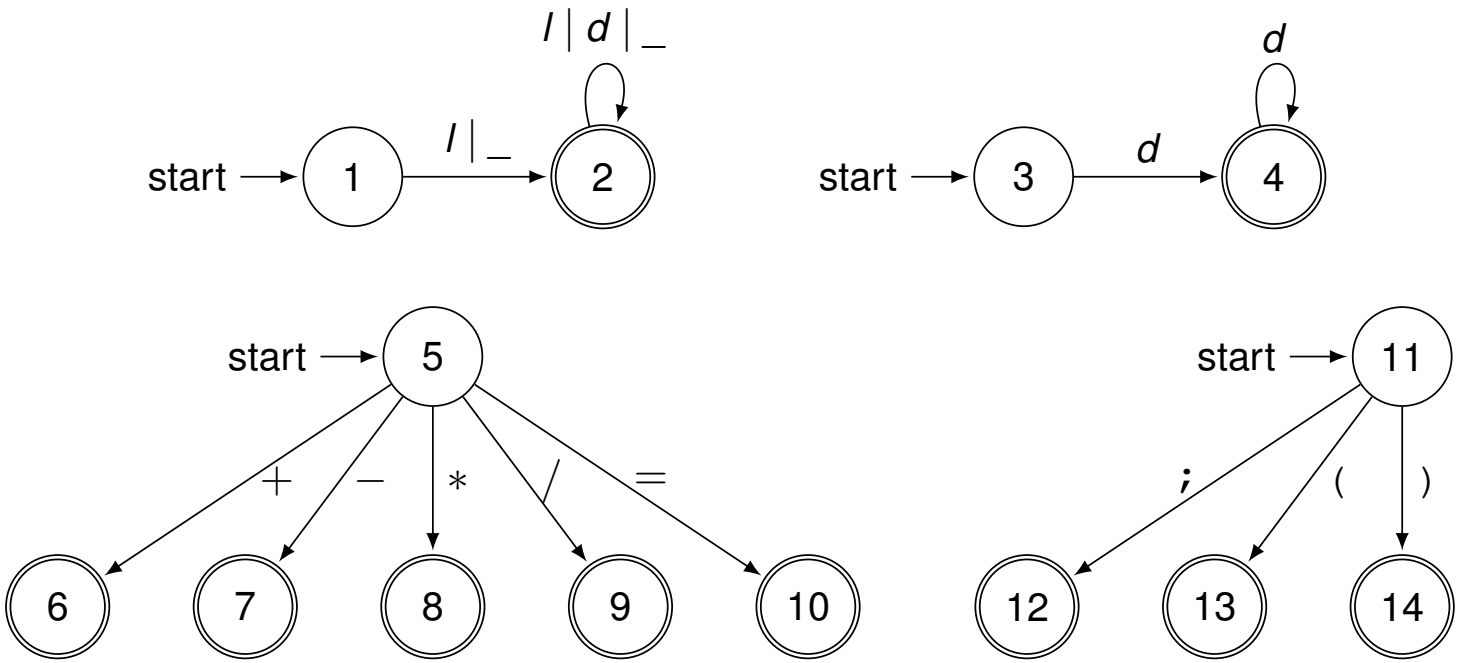
토큰표token table

- 문법을 통해 언어에서 사용되는 토큰들을 찾아 아래와 같이 토큰표를 만든다.

토큰 이름	토큰 번호	토큰 값
식별자	2	기호표 주소값
상수	4	상수 값
+	6	~
−	7	~
*	8	~
/	9	~
=	10	~
;	12	~
(	13	~
)	14	~

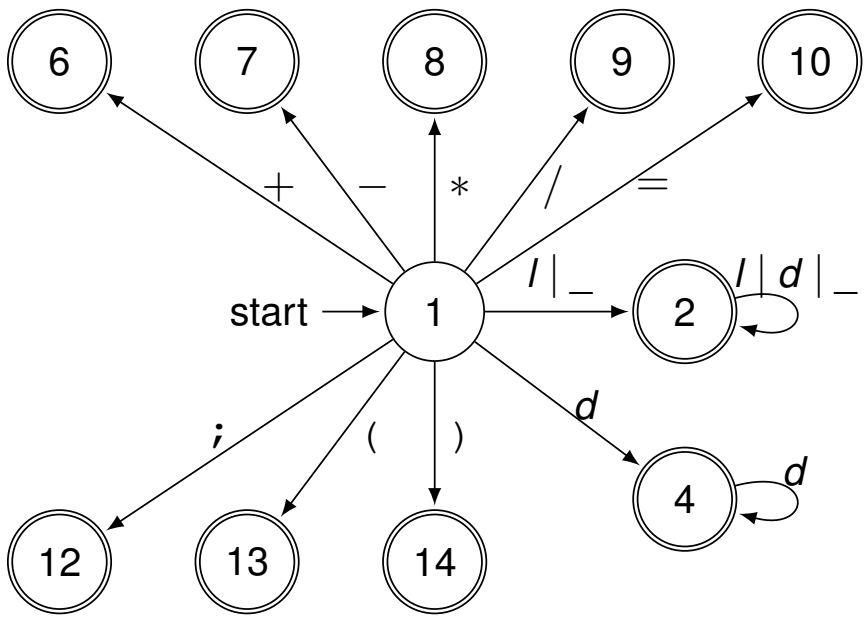
FA 구성

- 토큰표의 토큰들을 인식하기 위해 정규 표현식으로부터 FA를 구성한다.
- 아래는 각각 식별자, 정수, 연산자, 구분자를 인식하기 위한 FA이다.



FA 구성

- 토큰표의 토큰들을 인식하기 위해 정규 표현식으로부터 FA를 구성한다.
- 아래는 각각 식별자, 정수, 연산자, 구분자를 인식하기 위한 FA이다.



여러 개의 정규 표현식을 하나의 최소화된 DFA로 변환한 결과

어휘 분석기 코드 예시

- 아래는 Java로 구현된 간단한 어휘 분석기 코드이다.

```java
import java.util.Scanner;

public class LexicalAnalyzer {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        char[] string = new char[100];

        String stdin = input.nextLine();
        for (int j = 0; j < stdin.length(); j++) {
            string[j] = stdin.charAt(j);
        }

        System.out.println(string);
```

```
16          for (int i = 0; string[i] != 0; i++) {
17              if (string[i] >= '0' && string[i] <= '9') {
18                  System.out.print("Constant : " + string[i]);
19
20                  while (true) {
21                      i++;
22                      if (string[i] < '0' || string[i] > '9') {
23                          break;
24                      }
25                      System.out.print(string[i]);
26                  }
27                  System.out.println();
28                  System.out.println("token num : 4 ");
29              }
30
31              if (string[i] == '+') {
32                  System.out.println("Operator : + ");
33                  System.out.println("token num : 6 ");
34              }
35              ...
36          }
37      }
38  }
```

Any questions?

참고문헌

- A. Aho, J. Ullman, R. Sethi, M. S. Lam, Compilers: Principles, Techniques, and Tools (2nd Edition), Addison Wesley, 2006
- R. Sebesta, Concepts of Programming Languages, 5th Edition, Addison-Wesley, 2001
- K. C. Louden, Compiler Construction: Principles and Practice, Cengage Learning, 1997
- K. C. Louden and K. A. Lambert, Programming languages : Principles and Practice, 3rd Edition, Cengage Learning, 2012
- 박두순, 컴파일러의 이해, 한빛아카데미, 2016
- 김종훈, 김종진, 프로그래밍 언어론 : 쉽게 배우는 언어의 원리와 구조, 한빛미디어, 2013