5. 어휘분석

충북대학교

이 재 성

LaKE LAB

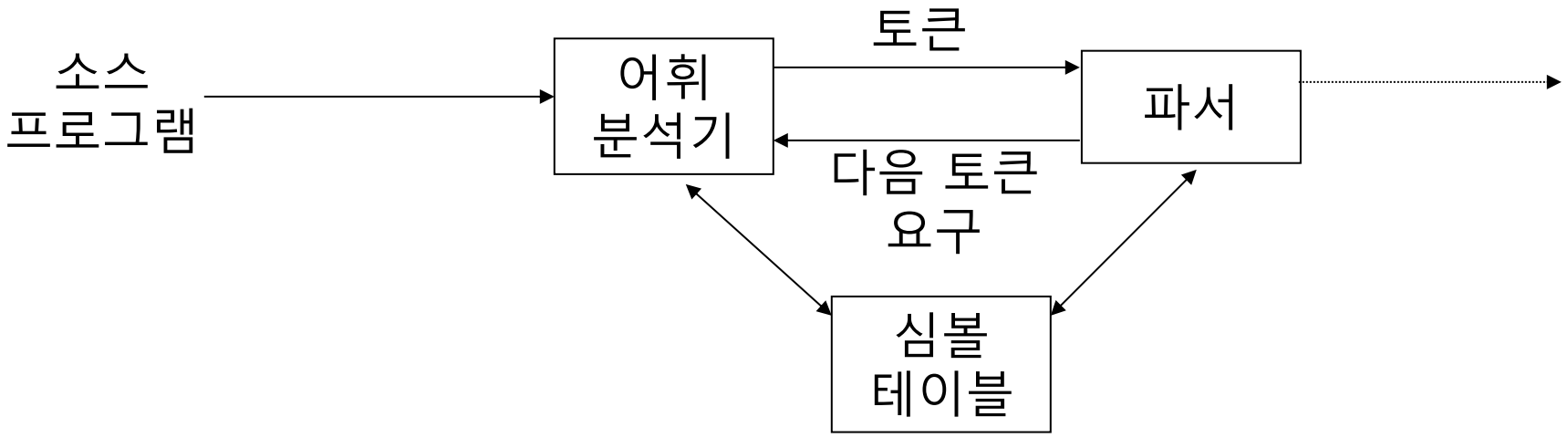
학습내용

- 어휘 분석기 기능 및 구현
- 언어 및 정규 표현
- 전이다이어그램
- 전이다이어그램을 이용한 어휘분석기

어휘분석기의 역할

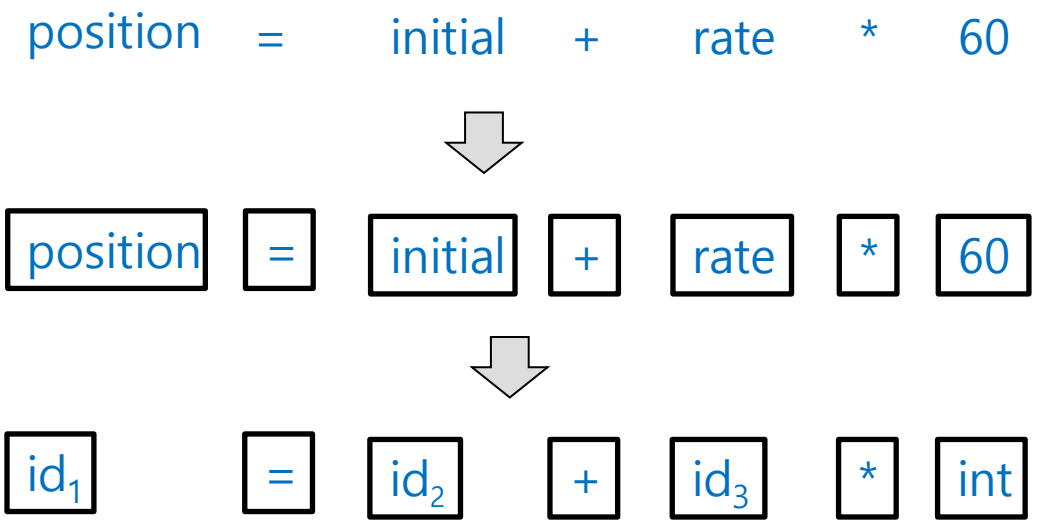
역할

- 주요 역할:입력 문자열을 읽어 토큰의 생성
- 부차적 역할: 빈공간 제거, 오류메시지와 소스 프로그램 연관 (오류줄)



어휘 분석 기능 (1)

- 선형 분석 (스캐닝)
- 문장을 토큰 단위로 분리



심볼테이블

idx	name	token
1	position	id
2	initial	id
3	rate	id

어휘분석기 기능 (2)

식별자와 키워드의 인식

- 키워드
  - 프로그램언어에서 특수 의미를 갖는 문자열(for, if, while, int 등)
  - 예약어와 식별자 구분
- 심벌테이블 이용
  - 식별자의 실제 문자 값(렉심)과의 연결
  - 파서에는 실제 문자열 값을 속성값으로 넘겨줌
  - 렉심(어휘소, lexeme)
    - 하나의 토큰을 만들어내는 연속적인 입력 문자들

상수 처리

- 토큰 번호와 함께 상수의 실제 값을 속성 값으로 넘겨줌

어휘분석기 기능 (3)

- 여백과 주석(설명)문 제거
- 입력버퍼 처리
  - 토큰을 만들기 전에 다음 글자를 읽어야 할 경우가 있음
  - 예: '>' 다음에 한 글자를 읽고, 그 글자가 '=' 일 경우는 '>='를 토큰으로 넘기고 그 이외에는 다음 읽은 글자를 다시 버퍼로 두고 처리
  - 되돌리기(retraction) 기능에 필요 (ungetc()함수 이용 가능)
- 토큰 번호 부여
  - 파서와의 통신
  - 일반적으로 글자를 그 코드 값 그대로 넘겨주는 경우가 있으므로 숫자나 식별자와 같은 토큰 값은 255보다 큰 값들을 부여

if (A > B) 에서 if 뒤에 문자가 있는지 없는지 체크하고 뒤로 다시 가는걸 retraction 이라고 한다.

어휘분석과 파싱의 분리

어휘분석의 분리목적

- 간단한 설계
  - 주석이나 빈 공간이 미리 처리되어 파싱에서 신경 쓸 필요가 없음
- 컴파일의 효율향상
  - 대상 파일에서 토큰을 읽기 위해 특수 버퍼링 기법으로 속도 단축
- 이식성 증가
  - 토큰은 문자가 아닌 숫자로 파서에 전달
  - 따라서 토큰을 어휘분석에서 인식만 하면 언어 독립적인 처리 가능

토큰, 패턴, 렉심

패턴(pattern)

- 동일한 토큰값을 갖는 문자열의 집합

렉심(lexeme)

- 패턴에 적합한 문자들의 나열

토큰	렉심의 예	패턴의 비공식적 설명
for	for	for 문자열
if	if	if 문자열
*	*	* 문자
relation	<, <=, = ...	< 또는 <= 또는 = 등...
id	pi, count, D2	문자로 시작되는 문자와 숫자의 열
num	5, 3.14, 6.1e23 ...	임의의 수치 상수

토큰 구분 요소

- 자유입력 형식
  - 구성요소가 라인의 어느 위치에 오든 관계없이 처리: 예 C언어
  - cf: Fortran - 고정위치
- 공백의 처리
  - 공백의 무시: 토큰 식별의 어려움
  - 예: Fortran에서 DO 5 I=1.25와 DO 5 I=1,25

ex) sum x = 5 -> 오류 발생
sumx = 5 ㄱㅊ

- 예약어 처리
  - 문자열이 키워드로 예약되지 않으면 처리 복잡
  - 예: PL/I
    - IF THEN THEN THEN = ELSE; ELSE ELSE = THEN;

심볼 테이블

소스 언어의 정보유지

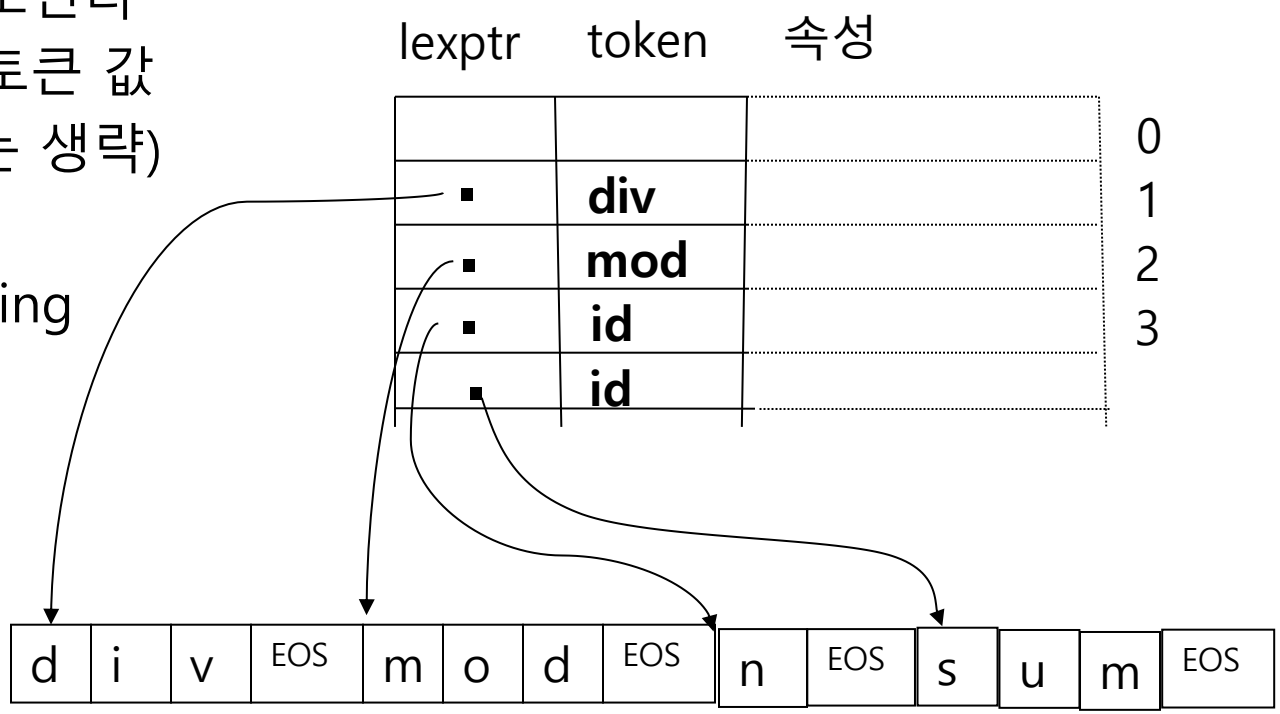
- 렉심의 저장 및 검색
- 구현 함수 예
  - insert(s, t): 새로운 문자열 s와 토큰 t를 저장하고 그 위치를 돌려줌
  - lookup(s): 문자열 s를 찾아서 그 위치를 돌려줌 (없으면 0)

심볼 테이블을 이용한 예약어 처리

- 예약어를 심벌테이블에 토큰값과 함께 저장
- 예: insert(“div”, div); insert(“mod”, mod);

심볼 테이블 구현 예

- 심볼 테이블 구성
  - lexptr: 렉심의 포인터
  - token: 렉심의 토큰 값
  - 속성(여기에서는 생략)
  - EOS: end of string



어휘분석기 가상코드

```
loop begin
        c에 한글자 읽기
        if c가 공백 혹은 탭 then 무시
        else if c가 개행문자 then lineno++;
        else if c가 숫자 then
                c와 그 뒤의 숫자의 값을 tokenval에 대입
                return(NUM);
        else if c가 문자 then
                c와 그 뒤의 문자 혹은 숫자들을 lexbuf에 대입
                p = lookup(lexbuf);
                if p==0 then
                        p= insert(lexbuf, ID);
                tokenval = p;
                return 테이블 엔트리 p의 token값
        else /* 하나의 문자가 토큰으로 가정 */
                tokenval에 NONE대입
                return c
end /* loop */
```

정규표현 및 전이다이어그램과 어휘분석기

- 정규표현
  - 문자열 패턴 정의 방법
  - 어휘분석기의 토큰 패턴 정의
- 전이다이어그램(상태전이도)
  - 정규표현을 다이어그램으로 표현
- 어휘분석기
  - 전이 다이어그램을 프로그램화

정규표현 ⇒ 전이다이어그램 ⇒ 어휘분석기

문자열과 언어

정의

- 문자열: 알파벳에서 뽑아낸 기호들의 유한순서
  - 문자열 s의 길이 표시: |s|
  - 빈 문자열 표시: ε
- 언어: 고정된 알파벳으로 만들어진 문자열들의 집합
- 문자열의 연결
  - x와 y의 연결 xy
  - 곱으로 표현
    - $s^0 = \varepsilon$로 정의
    - $s^1=s$, $s^2=ss$, $s^3=sss$
- 문자의 부분열
  - 문자열의 일부(단, ε나 자기 자신도 부분열)

언어 연산 - 1

합집합(union)

- 언어 L과 M의 합집합 표시는 L ∪ M
- L ∪ M = {s | s는 L또는 M의 요소}

연결(concatenation)

- LM으로 표시
- LM = {st | s는 L의 요소이고 t는 M의 요소}

언어 연산 - 2

클로저(closure)

- 클리네 클로저
  - L*로 표시
  - $L^* = \bigcup_{i=0}^{\infty} L^i$
- 양의 클로저
  - L+로 표시
  - $L+ = \bigcup_{i=1}^{\infty} L^i$

예) L이 알파벳, D가 한자리 숫자를 나타낼 경우,

L(LUD)*는 식별자

D+는 숫자를 나타냄

L = { a, ... , z A, ... , Z }
D = { 0, ... ,9 }

NUM = { D + }

ID => L (L U D)*

정규표현(regular expression) - 1

정의 규칙

- ε는 {ε}를 나타내는 정규표현
- 만약 a가 S에 속한 기호이면 a는 {a}를 표현하는 정규표현
- r과 s가 정규표현 L(r)과 L(s)를 나타낸다고 할 때,
  - (r)|(s)는 L(r) U L(s)를 나타내는 정규표현
  - (r)(s)는 L(r)L(s)를 나타내는 정규표현
  - (r)*는 (L(r))*를 나타내는 정규표현
  - (r)은 L(r) 를 나타내는 정규표현

정규표현(regular expression) - 2

불필요한 괄호를 없애기 위한 규약

- 단항연산자 *는 가장 높은 우선순위를 가지면 좌측 연관적
- 연결은 두번째 우선 순위를 가지며 좌측 연관적
- |는 가장 낮은 우선 순위를 가지며 좌측 연관적

정규집합

- 정규표현으로 표현되는 언어

정규표현의 대수

- 교환법칙
  - r|s = s|r
- 결합법칙
  - r|(s|t) = (r|s)|t
  - (rs)t = r(st)
- 배분법칙
  - r(s|t) = rs|rt
  - (s|t)r = sr|tr
- 단위원
  - εr = r
  - rε = r
- 기타
  - r* = (r|ε)*
  - r**=r*

정규표현의 축약 표기

축약표기 연산자

- 단항 후위 연산자 +
  - 하나 또는 그 이상의 개수가 나타남
- 단항 후의 연산자 ?
  - 0개 또는 하나가 나타남
- 문자클래스
  - [abc]는 정규표현 a|b|c를 표현
  - [a-z]는 정규표현 a|b|...|z를 표현

a+ = { a, aa, aaa, ... }
a* = { SIGMA, a, aa, ... }
ba?g = { bg, bag }

a가 들어갈 수도 있고 아닐 수도 있고...

e.g)
a+b?c

1. ac 2. bc. 3. abc 4. acd

속하는 것: { ac, abc }

e.g2)
ID = L (L|D)*
NUM = D+

letter = [a - z A - Z]
digit = [0 - 9]

[abc]x*z

1. bxz 2. cz 3. abz 4. xxxz

속하는 것: { bxz, cz }

원래 ID, NUM 등에서 표현하는 식은 하나하나 {} 중괄호로 씌워줘야 하지만, 편의상 이렇게 적어도 무관하다.
만약 겹치는게 있다면, 씌워주는 것이 좋다.

전이 다이어그램

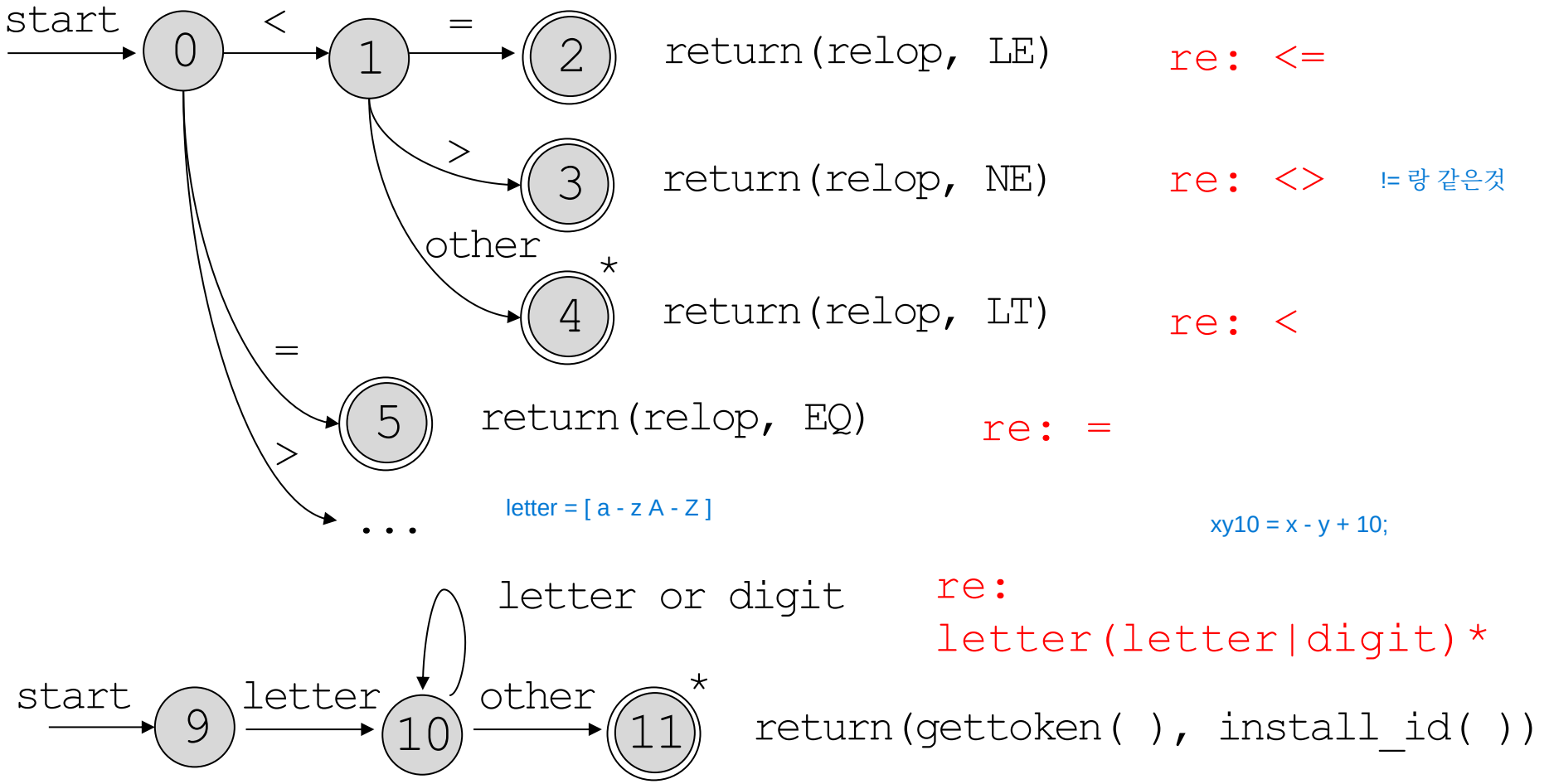
역할

- 정규표현을 다이어그램으로 표시하여 어휘분석기로 구현토록 함

구성

- 상태(state) - 다이어그램의 각각의 위치이며 원으로 표현
- 연결선(edge)
  - 다음에 나타날 수 있는 입력문자의 레이블을 표시한 화살표
  - "other" 레이블은 다른 연결선에서 지칭하지 않고 s를 떠나는 임의 문자
- 시작상태 - start라고 이름이 붙여진 다이어그램의 초기 상태
- 도달상태 - 토큰이 발견된 상태로 이중원으로 표시
- 입력의 되돌리기(retraction) - 토큰이 아닌 부분까지 읽었을 경우 되돌리는 표시로 상태 위에 *로 표시

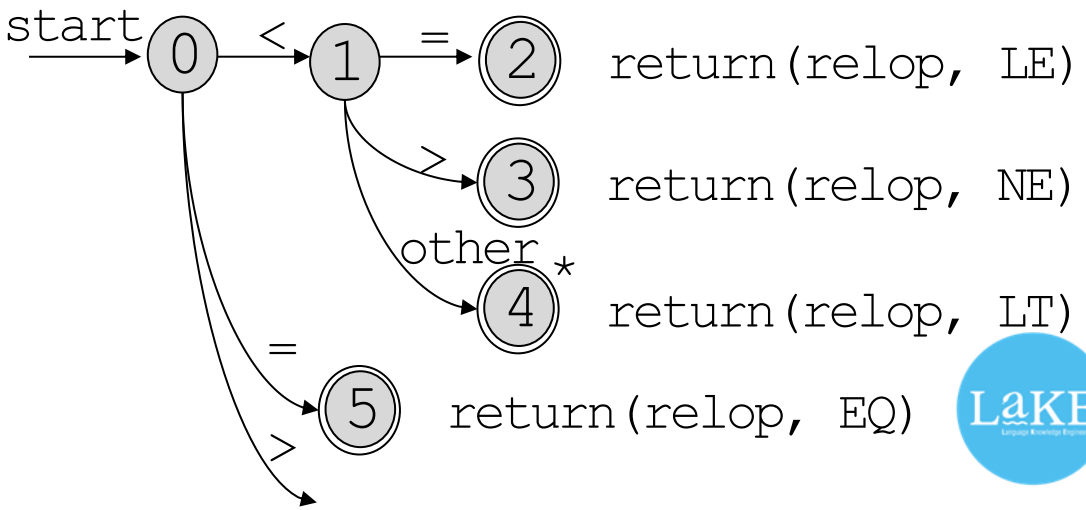
전이 다이어그램 예


start
0
<
1
=
2
return(relop, LE)
re: <=
>
3
return(relop, NE)
re: <>
!= 랑 같은것
other
*
4
return(relop, LT)
re: <
=
5
return(relop, EQ)
re: =
>
...
letter = [a - z A - Z]
xy10 = x - y + 10;
letter or digit
re:
letter(letter|digit)*
start
9
letter
10
other
*
11
return(gettoken( ), install_id( ))

전이 다이어그램을 어휘분석기로 구현(예)

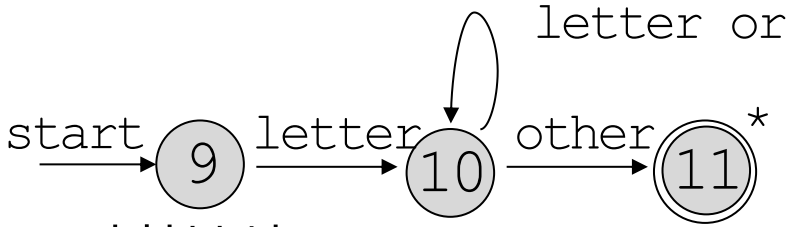
```
token nexttoken()
{ while(1) {
switch(state) {
case 0: c = nextchar();
        if (c==blank || c==tab || c==newline){
                state = 0; lexeme_beginning++;}
        else if (c == '<') state = 1;
        else if (c == '=') state = 5;
        else if (c == '>') state = 6;
        else state = fail();
        break;
        ...
```

ungetc() 중요!



전이 다이어그램을 어휘분석기로 구현(예)

```
        :
        case 9: c=nextchar();
                if (isletter(c)) state=10;
                else state=fail();
                break;
        case 10: c=nextchar();
                if (isletter(c)) state=10;
                else if (isdigit(c)) state=10;
                else state=11;
                break;
        case 11:
                retract(1); install_id();
                return(gettoken());
        :
```



return(gettoken( ), install_id( ))

참고 문헌

- [1] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, “Compilers – Principles, Techniques, and Tools,” Bell Telephone Laboratories, Incorporated, 1986.
- [2] 오세만, “컴파일러 입문”, 정익사, 2004.