

## 6주차 Lexical

2022년 4월 6일 수요일 오후 11:44

### Lexical Analysis (Scanner)

★ 전체적인 개념 및 연습문제 포함

★ 지금까지의 내용 정리 한 느낌

어휘 분석기



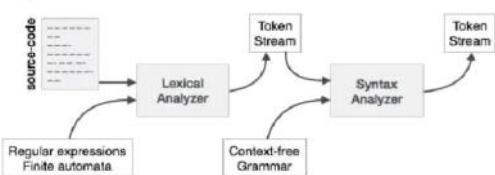
화면 캡처: 2022-04-06 오후 11:49

Source program에서 token을 추출하는 구문을 의미한다

Lexical analyzer는 token을 분리하고 parser에게 넘긴다  
실제는 parser가 token 필요할 때마다 get\_token() 한다



화면 캡처: 2022-04-07 오전 11:43



Parser는 token을 이용한 parse tree build한다.

Parser는 token 필요하면 lexer를 통해 처리를 한다 -> 가볍다. 빠르다.

### Lexical module

lexical은 parser의 sub로 자리 잡고 있다

#### parser(syntax analysis)와 scanner(lexical analysis) 구분이유

1. Token은 Regular languages이다.
  - a. Regular languages는 context free language보다 간단한 방법으로 정의할 수 있다
  - b. Lexier 즉 정규언어 인식기는 parser와 분리하여 쉬운 방법으로 구현 할 수 있다. => 컴파일러 front-end의 모듈화 = Lexer + parser
2. Lexical 하는 일 parser가 다 할 수 있지만 모듈화 문제, token 구문구조 정의가 합쳐지는데서 오는 문제, 프로그램이 커지는 문제

### Token

문장에서 사용되는 최소 단위의 문법 요소 - **Terminal symbol**

예) if ( x > y ) x = 10 ;  
(32,0) (7,0) (4,x) (25,0) (4,y) (8,0) (4,x) (23,0) (5,10) (20,0)

화면 캡처: 2022-04-07 오전 11:17

(Token name, Token value)

#### Token 정의

- Keyword
  - Constants
  - Identifiers
  - String,integer,float,double...
  - Operator ...
- int (keyword), value (identifier), = (operator), 100 (constant) and ; (symbol).

Regular Expression으로 표현합니다.

Category	Regular Expression
Keyword	bool   char   else   false   float   if   int   main   true   while
Identifier	letter (letter   digit)*
integerLit	digit*
floatLit	digit . digit *
charLit	'anyChar'

### Category Regular Expression

Operator	=   +   -   &&   ==   !=   <   <=   >   >=   *   /   !   [ ]
Separator	:
Comment	// (anyChar   whitespace)* col

화면 캡처: 2022-04-07 오전 11:58

- IDENTIFIERS
  - a,b,c,sum... ID(모든 종류의 식별자)
- LITERALS
  - 123,'x',true,false
- KEYWORDS
  - Int , float, double, ..
- OPERATORS
  - +\*/-
- PUNCTUATION
  - ;,(),.

추가로 Lexier가 하는 역할

Lexical Analyzer는 parsing하는데 필요 없는 것들은 버립니다. (예) whitespace, comment

1. whitespace 제거
2. Comment제거
3. Line counting
  - a. 몇번째 라인인지 계산한다 line comment 처리하기 위해 별도로 구분 처리

### Source program

```
// A program with  
// two line comments  
int main(){  
    char c;  
    int i;  
    c = 'h';  
    i = c + 3;  
} // main
```

### Tokens

int
main
(
)
{
char
Identifier
c
;
}

화면 캡처: 2022-04-07 오전 11:45

Regular languages

정의 방법

정의하기 쉬운 것은 terminal Symbol(非-terminal Symbol) 구조

### 1. Regular grammar

Regular grammar(RG).

$A \rightarrow tB$  or  $A \rightarrow t$ , where  $A, B \in V_{NT}$ ,  $t \in V_T^*$ .

$A \rightarrow tB$  or  $A \rightarrow t$  이라는 구조로 Regular Grammar

화면 캡처: 2022-04-07 오전 11:48

t는 terminal A,B는 non terminal symbol이다

EX 1) 010으로 시작하는 모든 binary string 생성하는 문법

S-> 010A

A->0A | 1A | 입실론

| 010은 terminal A는 non-terminal  $A \rightarrow t$  구조

이때 010은 Terminal symbol이고 A는 non terminal symbol이다

EX 2)

1. 영문자 또는 '\_'로 시작하여 영문자와 숫자로 구성되는 변수 이름  $A \rightarrow tB$  구조
2. 모든 삼진수 정수  $A \rightarrow t$  구조
3. JAVA 의 연산자 집합

모두 regular grammar를 이용하여 regular language로 정의할 수 있다

4. 유한개의 string으로 이루어진 집합은 정규 언어이다

i. YES

ii. RG를 이용해서 만들 수 있는 언어, 모든 알파벳, 숫자는 정규언어니까

|  $A \rightarrow t - \rightarrow A \rightarrow t$  구조로 만들 수 있다.

### ★2. Regular expression

Regular Expression	Language
$\epsilon$	{ $\epsilon$ }
a, where $a \in T$	{a}
$P \cup Q$	$L_P \cup L_Q$
$PQ$	$L_P L_Q$
$P^*$	$L_P^*$

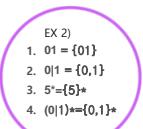


☞ ACT 애야~한다.

화면 갱신: 2022-04-07 오전 11:54

$P, Q$ 는 각각 RE이다.  
Regular Expression을 결합해서 새로운 RE 정의 할 수 있다

EX)  
011으로 시작하는 바이너리 스트링의 집합  $011(0|1)^*$   
011으로 끝나는 바이너리 스트링의 집합  $(0|1)*011$



Regular Expression 예제들

문헌에 따라 |를 ↗로 표기하기도 한다

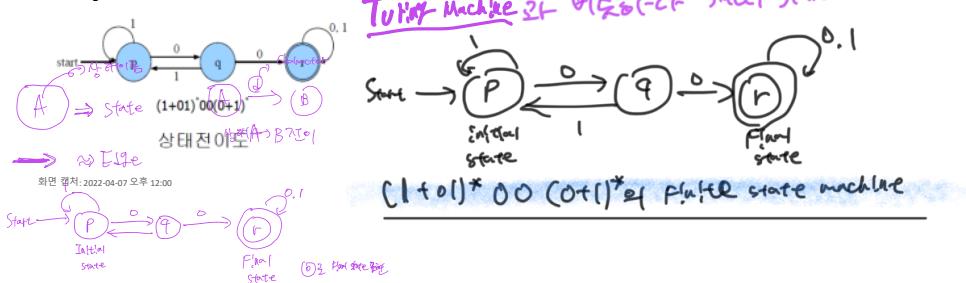
### 3. Finite automata Regular Expression으로 이어진 Token을 판별할 수 있게 해주는 보드

주어진 input string x에 대해  $x \in L$ 일 때 YES라고 판정하는 기계 (turing machine과 비슷하다)

regular expression을 인식하기 위해 존재 한다

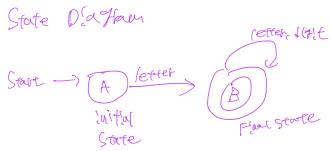
) 실행자 및 적어도 하나의

State Diagram



Letter (Letter | Digit)\* ID or keyword token 만들려는 Regular Expression을  
plate state machine이 어떤가 표현하나?

Scan 234 인식하는 Finite state machine



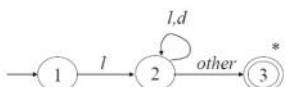
Scan 234

↳ 반드시 키워드는 Token에 포함되지 않는 Symbol을 선택한다.



lexer 구현 하는 과정에서도 빙컨을 기준이 되어서 token에 포함되지 않는 symbol 있어서 구분 했다.

마지막 읽은 걸 다시 Push back 해준다.



State diagram

상태	입력	letter	digit	other
1	2	error	error	
2	2	2	3 (accept)	

State transition table

## Lexer 구현

모든 방식을 위의 구조로 만들면 복잡해진다  
그래서 한 언어의 automata는 종류별로 token 인식하는 sub-automata들을 통합하여 구성할수 있다

시작 상태에서 출발하여 token 한 개를 인식하고 다음 token을 인식하기 위해 다시 시작 상태로 돌아간다  
이때 돌아가기전에 다음 토큰에 사용되는 글자를 push back하고 돌아가야함

-> 입력에서 토큰의 끝을 확인하기 위해서는 그 토큰 다음에 오는 글자를 하나 더 읽어야한다

이 글자는 다음 토큰의 일부이므로 다음 토큰의 인식을 위해 시작 상태로 돌아가기 전에 읽지 않은 상태로 되돌려 놓아야한다 pushback

## Push back 방법 2가지가능

- Push back function 구현
  - C에 ungetch()

char ch;

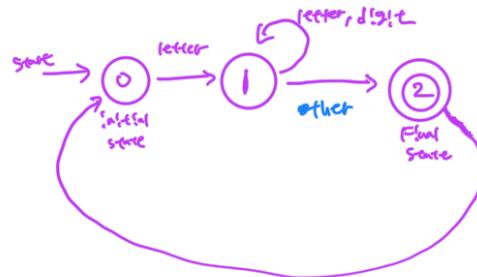
```
char getNextChar() {
    char c;
    if (buffer != NULL) {
        c = ch; ch = NULL;
        return c;
    }
    else
        return getch();
}
```

void pushback(char a) { ch = a; }

pushback 함수의 구현(getNextChar())

Ex) Sum 234 = a \* (2);

letter (letter + digit)\*



=를 인식하고 Push back (=)

## 2. 시작 상태로 돌아오기 전에 항상 한 글자 미리 읽어 두기

## 방법 2 사용하기(단계)

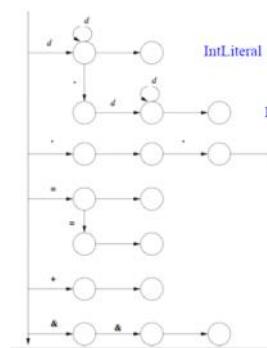
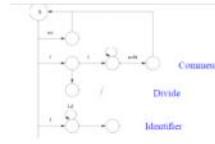
## 방법 2는 가능

시작 상태에는 항상 한 글자가 버퍼에 미리 읽혀져 있다고 가정한다. 즉, 첫 글자는 버퍼에서 읽는다.

- 따라서, 모든 sub-automata에서 토큰을 인식한 후에는 항상 다음 글자를 버퍼에 읽어놓고 시작상태로 돌아간다.

Automata가 첫 토큰을 인식할때는 어떻게 할까?

-> 토큰 인식에 해가 되지 않는 아무 문자 삽입해서 읽어옴 ex.) 빈칸



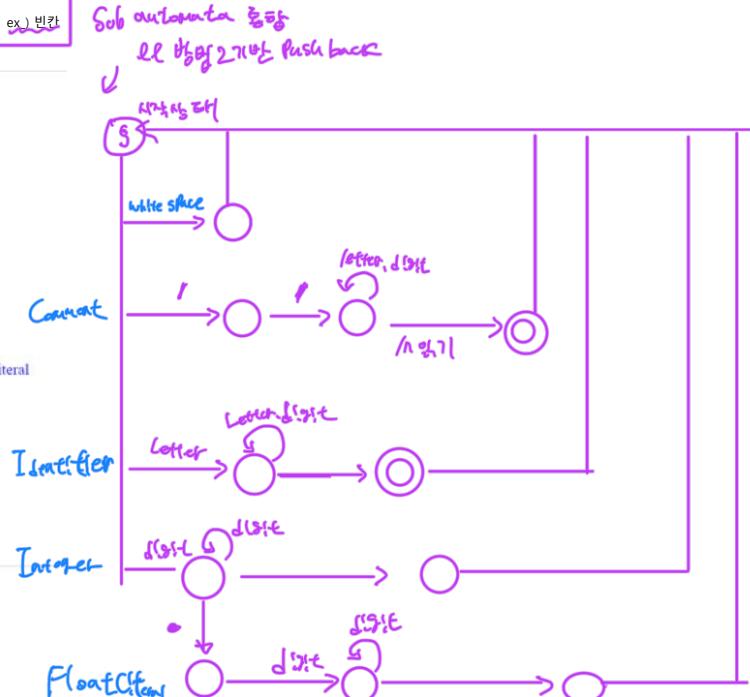
## 3. Transition from one state to another

## 1. 시작 상태인 경우

- 버퍼 ch에 저장된 입력과 일치하는 arc를 따라서 다음상태로 이동한다. 일치하는 arc가 없다면 error

## 2. 시작 상태가 아닌 경우

- Read the next character into ch
- If ch와 일치하는 arc가 있다면
  - String에 글자를 추가하고 다음상태로 이동
- If there is an unlabeled arc
  - Read a char into ch and follow that arc
  - Token인식 완료
- Error
  - Error



## From Design to code

```
Public Token next()
{
    do
    {
        If(isLetter(ch))
        {
            String spelling = concat(letters+digits);
            return Token.keyword(spelling); //identifier, keyword 같이 받아서 마지막에 구분한다
        }
        Else if(isDigit(ch))
        {
            String number = concat(digits);
            If(ch!='.')
            {
                Return Token.mkIntLiteral(number); //integer일때 반환
            }
            Number+=concat(digits);
            Return Token.mkFloatLiteral(number); //float일때 반환
        }
        Else switch(ch)
        {
            Case ' ': case "WT" : case : 'Wr' case eolCh:
                Ch = nextch();
                Break;
                ....
        }
    }
}

}while(true)

Private bollen isLetter(char c)
{
    Return ch>='a' && chM<="z" || ch>="A" && ch<="Z";
}

Private String concat(String set)
{
    StringBuffer r = new StringBuffer("");
    Do
    {
        r.append(ch);
        Ch = nextChar();
    }while(set.indexOf(ch)>=0);

    Return r.toString();
}
}
```

## 6주차 과제 토큰+ 프로그램 부연 설명

<https://it-q.com/how-are-lexical-errors-identified/>

과제 목표 : cli의 lexical analyzer C로 구현

### 토론

- 언어의 token을 regular expression으로 명세하라

Category	Regular Expression
Keyword	bool   char   else   false   float   if   int   main   true   while
Identifier	letter (letter   digit)*
integerLit	digit*
floatLit	digit *.digit *
charLit	'anyChar'

### Category Regular Expression

Operator	=   +   -   &&   ==   !=   <   <=   >   >=   *   /   !   [   ]
Separator	:
Comment	// (anyChar   whitespace)* eol

- Token을 인식하는 Finite State Machine을 구현하라

3. Source program에 오류가 있을 경우 어떻게 하는 것이 좋을까 생각해보라.

## 1. Lexical level과 Syntax level의 오류는 어떤 차이가 있는가?

예시

```
int 2sb;  
int 2;  
Lexical error는 compiler가 적절한 token의 sequence of character인가를 인지 못할때 발생  
Valid token 만들수없을때  
2ab는 변수 규칙에 맞지 않는다
```

Syntax error는 int 2처럼 문법이 맞지 않을때 발생  
Token이 해당 문법에 맞지 않을때 발생

## 1. Compiler가 만나는 오류의 종류를 level 별로 분류할 수 있는가? 그 종류를 나열해보라.

- a. Syntax error-> 문법 오류
- b. Semantic error-> 프로그램은 돌아가는데 의도한대로 나오지 않는 오류
  - i. 논리적으로는 맞는데 변수를 다른것을 썼다거나등의 이유로 원하는 결과가 나오지 않는점에서 논리적 오류와 차이가 있다
- c. Lexical error

## 2. Lexer는 syntax level의 오류를 어떻게 처리하는 것이 좋을 지 생각해보라.

- a. Parser에서 일은 문자열을 token으로 바꾸어서 syntax error 처리 한다
- b. Lexical은 문자가 잘못 되었거나 이런건 detect할수있지만 문법적으로 틀린것인지는 탐지 할수없느느데 그래서 source를 token으로 바꾸어서 parse tree만들어서 syntactical error를 탐지한다?

Parser의 목적

1. 모든 syntax error를 발견하고 적절한 진단 메시지를 제공하여 error 해결하는것
2. 프로그램에 parse tree를 제공하는것

## Lexical Error (Scanner error)

<https://stackoverflow.com/questions/3484689/what-is-an-example-of-a-lexical-error-and-is-it-possible-that-a-language-has-no>

character의 순서가 Token pattern과 맞지 않을때 발생한다  
Token pattern과 일치하는지 본다

A pattern explains what can be a token, and these patterns are defined by means of regular expressions.

출처: <[https://www.tutorialspoint.com/compiler\\_design\\_compiler\\_design\\_lexical\\_analysis.htm](https://www.tutorialspoint.com/compiler_design_compiler_design_lexical_analysis.htm)>

Regular expression is an important notation for specifying patterns. Each pattern matches a set of strings, so regular expressions serve as names for a set of strings. Programming language tokens can be described by regular languages. The specification of regular expressions is an example of a recursive definition. Regular languages are easy to understand and have efficient implementation.

1. Signed integer는 범위 안에 있어야하는데 범위 밖에 숫자를 쓸수있을때?
2. printf("Geeksforgeeks");과 같이 illegal character \$가 있는 경우

A **syntax error** occurs when you write a statement that is not valid according to the grammar of the C++ language. This includes errors such as missing semicolons, using undeclared variables, mismatched parentheses or braces, etc...





