6. LEX

A Lexical Analyzer Generator

충북대학교

이 재 성

LaKE LAB

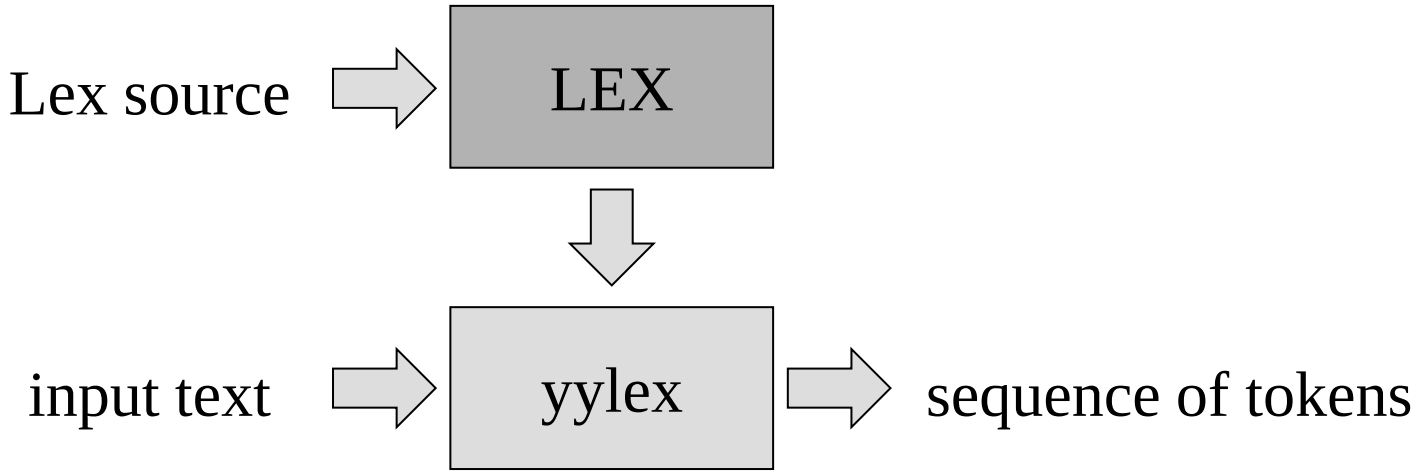
학습내용

- 어휘분석기를 자동생성하는 도구 Lex
- Lex의 소스 작성 방법
- 간단한 Lex 소스 예

소개

- Lex
  - 입력된 정규표현을 프로그램으로 변환

- Lex의 역할



(1) Lex는 사용자의 정규 표현과 명령을 프로그램 언어로 변환한다.

Lex source : *.l

(2) yylex 함수는 입력 문자열에서 정규표현을 인식하고 발견된 각 정규 표현에 해당되는 특정 명령들을 실행한다.

Lex 소스 형식

- 형식:

```
definitions                 // 정의
%%
rules                       // 규칙
%%                          // 이하 생략 가능
user routines               // 사용자 부프로그램
```

Lex 사용자 부프로그램

definitions
%%
rules
%%
user routines

- 사용자 부프로그램(user routines)
  - 규칙의 명령에서 사용되는 부프로그램들의 집합 정
  - Lex의 출력 후, lex.yy.c 끝 부분에 복사된다.

Lex 정의 (definitions)

definitions
%%
rules
%%
user routines

- Definitions
  ::= 선언부 + 마크로 정의
- 선언부
  - %{ %} 사이에 있는 코드
  - Lex에 의해 아무 처리 없이 lex.yy.c의 앞부분에 복사된다
- 매크로 정의
  - Lex 규칙의 정규표현에 사용할 표현을 미리 정의

    name translation
  - 매크로 정의의 사용 : {name}

ex)

```
D               [0-9]
L               [a-zA-Z]
%%
{L}({L}|{D})*   return IDENT;
```

Lex 규칙(rules)

definitions
%%
rules
%%
user routines

- 규칙
  - 문법 규칙을 표현
  - 규칙에 대해 처리해야 할 명령어 나열
- 규칙 구성
  - rules ::= 정규표현(regular expressions) + 명령(actions)

```
ex) integer   printf("found keyword INT");
    color     { nc++; printf("color"); }
    [0-9]+    printf("found unsigned integer : %s\n", yytext);
```

Lex 정규표현

Lex 정규 표현

::= text characters + operator characters

text character

- 비교되고 있는 문자열에서 상응하는 characters와 매치한다.
- 알파벳과 숫자들은 항상 text characters이다.

operator character(연산자) --- " [] ^ – ? . * + () $ / {} % <>

(1) " (double quote) --- 쌍따옴표 사이에 있는 모든 문자를 텍스트문자로 취급

ex) XYZ"++" <=> XYZ++

(2) \ (backslash) --- 한 개의 문자를 escape

ex) XYZ\+\+ <=> XYZ++

(3) [ ] — 문자들의 종류

(가) - (dash) — 범위를 표시

ex) [a-z0-9] 모든 소문자와 숫자를 포함한 character class를 표시한다.
[—+0-9] 두 개의 부호화 숫자를 매치한다.

(나) ^ (hat) — 부정이나 여집합을 표시

ex) [^a-zA-Z]는 영문자를 제외한 모든 문자를 나타낸다.

(다) \ (backslash) — 8진법의 escape처럼 문자를 escape

ex) [\40-\176] 아스키 값 40인 공백부터 176인 ~(tilde)까지 모든 인쇄 가능한 문자를 나타낸다.

(4) . --- 개행 문자를 제외한 모든 문자들을 나타낸다.

ex) "__".*는 __ 부터 한 라인의 끝까지

(5) ? --- 선택을 의미하는 연산자

ex) ab?c <=> ac 또는 abc

(6) * , + --- 반복 표현

a*는 a가 0번 이상 반복될 수 있음을 나타낸다.

a+는 한 번 이상 반복될 수 있음을 나타낸다.

ex) [a-z]+

[0-9]+

[A-Za-z_] [A-Za-z0-9_]* --- Identifier

(7) | --- 하나의 선택을 의미하는 연산자
ex) (ab | cd) 는 ab or cd.

(8) ^ --- 라인의 시작에서만 인식

(9) $ --- 오직 라인의 끝에서만 인식

(10) / --- 접미 문맥을 명시할 때 사용
ex) ab/cd ab 다음에 cd가 이어서 나타날 때만 ab가 토큰으로 처리된다.
ex) ab$ <=> ab/\n

(11) < > --- 시작 상태 표시

(12) { } --- 정의된 이름을 확장할 때 사용

lex 정규 표현 요약

x	문자 “x”.
“x”	x가 연산자일지라도 “x”.
\x	x가 연산자일지라도 “x”.
[xy]	문자 x 또는 y.
[x-z]	문자 x, y 또는 z.
[^x]	x가 아닌 특정 문자.
.	개행이 아닌 특정 문자.
^x	라인의 시작일 때의 x.
x$	라인의 끝일 때의 x.
<y>x	Lex가 시작 상태 y일 때 x.
x?	선택적인 x.

lex 정규 표현 요약

x*	x의 인스턴스, 0개 부터 n개까지 올 수 있음.
x+	x의 인스턴스, 1개 부터 n개까지 올 수 있음.
x | y	x 또는 y.
(x)	x.
x/y	y를 따르는 x.
{xx}	정의 부문으로 부터의 xx의 변형.

Lex 명령(actions)

- 정규 표현에 일치되는 문자열, 즉 토큰이 인식되었을 때 실행해야 할 행동
- default action
  - 인식되지 않은 모든 문자에 대해 실행되는 default action은 입력을 출력으로 그대로 복사
- null action – 입력을 무시하고 싶을 경우

  ex) `[\t\n] ;`

  구분자로 사용된 공백, 탭, 개행 문자를 입력에서 무시하고 싶을 때 처리하는 방법
- | (alternation)
  - 동일한 액션 코드의 반복적인 표기를 생략

```
ex) [\t\n ] ; <=>   " "  |
                    "\t" |
                    "\n" ;
```

전역 변수와 함수

Lex 명령 작성시 사용할 수 있는 변수 및 함수

(1) yytext : 정규 표현과 매칭된 실제 문자열

ex) [a-z]+ printf("%s", yytext);

(2) yyleng : 매칭된 문자열의 길이를 나타내는 변수

ex) yytext[yyleng-1] : 매칭된 문자열의 마지막 문자

(3) ECHO : 출력에 매칭된 문자열을 출력

ex) ECHO <===> printf("%s", yytext);

(4) yymore : 현재 매칭된 문자열의 끝에 다음에 인식된 문자열이 덧붙여지도록 하는 함수

(5) yyless(n) : n개의 문자만을 yytext에 남겨두고 나머지는 다시 처리하기 위하여 입력 스트림으로 되돌려 보내는 함수

(6) I/O routines

1) input() 입력 스트림으로부터 다음 문자를 읽는 함수

2) output(c) 출력 스트림으로 문자c를 내보내는 함수

3) unput(c) 다시 처리될 수 있도록 문자c를 입력 스트림으로 되돌려 보내는 기능을 하는 함수, input()에 의해 다시 읽혀진다.

(7) yywrap() : Lex가 입력 파일의 끝에 도달할 때 요청된다.

모호한 규칙

규칙 모호성

- 하나의 문자열이 여러 개의 규칙에 적용될 경우

해결

1) 가장 길게 인식할 수 있는 정규 표현을 우선한다.

2) 인식할 수 있는 토큰의 길이가 같은 경우, 먼저 기술된 정규 표현을 선택한다.

ex) integer keyword action;

[a-z]+ identifier action;

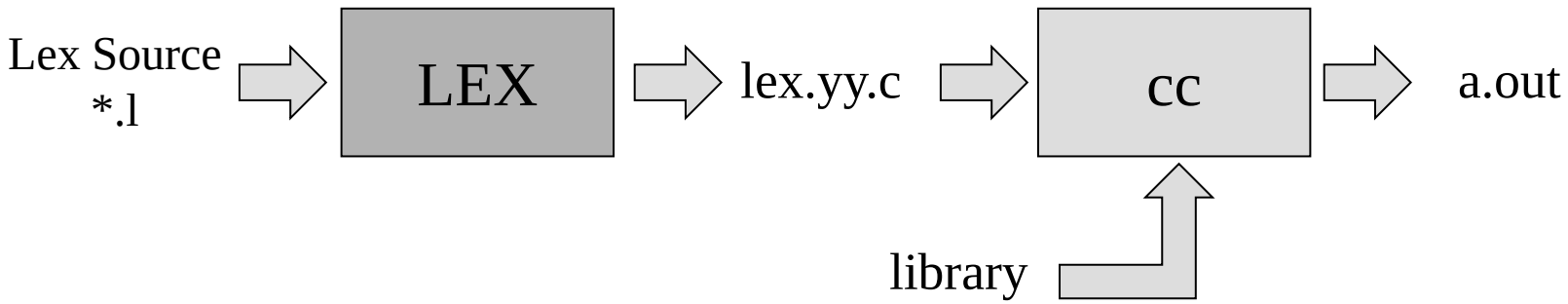
기본 동작 원칙

- Lex는 보통 각 표현의 가능한 모든 match를 찾는 것이 아니라 입력 스트림을 분할하는 것이다. 즉, 각 문자는 한 번만 처리한다는 의미이다.

Lex 소스 예제 (ex.l)

```
%{
#define  IF       100
#define  ID       101
#define  RELOP    102
#define           LE       201
int yywrap() {return(0);};
int yylval, tok;
%}
delim             [ \t\n]
ws                {delim}+
letter            [A-Za-z]
digit             [0-9]
id                {letter}({letter}|{digit})*
%%
{ws}              {/* do nothing */}
if                {return(IF); }
{id}              {yylval = install_id(); return(ID);}
"<="              {yylval= LE; return(RELOP);}
%%
int install_id() { return(1); }
main(){ while(1){ tok = yylex(); printf("%d %d\n", tok, yylval);}}
```

사용법



- PC용 Lex (flex):

```
C> flex -i -oexlex.c ex.l
C> cc exlex.c
C> exlex
a
101   1
if
100 1
```

참고 문헌

- [1] M.E. Lesk, A Lexical Analyzer Generator, Bell laboratories, Murry Hill, N.J. 07974, October, 1975.
- [2] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, "Compilers – Principles, Techniques, and Tools," Bell Telephone Laboratories, Incorporated, 1986.
- [3] 오세만, "컴파일러 입문" , 정익사, 2004.