정규표현
및
어휘분석기생성기 Lex

정규표현 (regular expressions)

알파벳 Σ:

- ε 은 정규표현이다. (언어 {ε}을 표현)
- ***a*** ∈ Σ 에 대해서, ***a*** 는 정규표현. (언어 {***a***}을 표현)
- ***r*** 과 ***s*** 가 정규표현이면 (각각 언어 L(***r***) 과 L(***s***)를 표현), 다음도 정규표현이다:
  - (***r***) | (***s***) (언어 L(***r***) ∪ L(***s***))
  - (***r***)(***s***) (언어 L(***r***)L(***s***))
  - (***r***)* (언어 L(***r***)*)

정규표현의 확장

- 한번 이상 반복: ***r+***
- 범위: **[a-zA-Z], [0-9]**
- 옵션: ***r?***
- 임의의 하나의 심볼: **.**
- 정규표현에 이름주기,

  e.g.:
  - letter = [a-zA-Z_]
  - digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  - ident = letter (letter | digit)*
  - Integer_const = digit+

정규표현의 예

식별자:

Letter = (a|b|c| … |z|A|B|C| … |Z)

Digit = (0|1|2| … |9)

Identifier = *Letter* (*Letter* | *Digit*)*

숫자:

[0-9]

[0-9]+

[0-9]*

[1-9][0-9]*

([1-9][0-9]*)|0

-?[0-9]+

[0-9]*\.[0-9]+

([0-9]+)|([0-9]*\.[0-9]+)

[eE][-+]?[0-9]+

([0-9]*\.[0-9]+)([eE][-+]?[0-9]+)?

-?(([0-9]+) | ([0-9]*\.[0-9]+)([eE][-+]?[0-9]+)?)

연습문제

❑ 문자열

[a-z]　　[a-zA-Z]　　[a-zA-Z0-9]

[a-zA-Z][a-zA-Z0-9]*

❑ 스트링

- "this is a string"

\".*\"　　<- wrong!!! why?

\"[^"]*\"

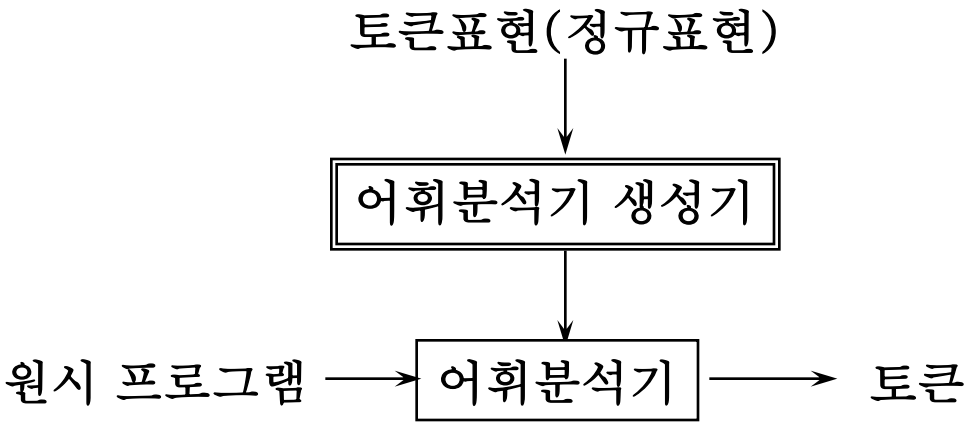
❑ 몇가지 연습

0과 1로 이루어진 문자열 중에서...

- 0으로 시작하는 문자열　　0[01]*
- 0으로 시작해서 0으로 끝나는 문자열　　0[01]*0
- 0과 1이 번갈아 나오는 문자열　　_____
- 0이 두번 계속 나오지 않는 문자열　　_____

어휘분석기 생성기

❑ 어휘분석기 생성기(lexical analyzer generator)의 개요



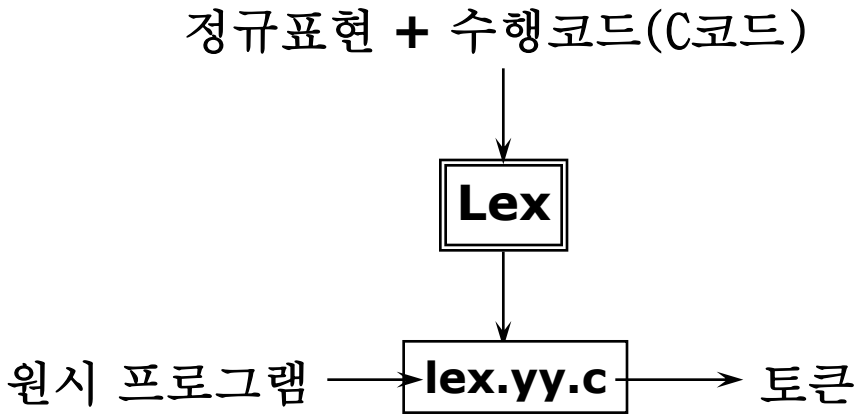
	입력	출력
어휘분석기	원시 프로그램	토큰
어휘분석기 생성기	토큰표현	어휘분석기

Lex

❑ Lex

- ❖ 1975년, Lesk와 Schmidt
- ❖ UNIX 운영체제에서 소프트웨어 개발도구로 개발

❑ Lex의 개요



Lex 실행 단계

- **Lex 프로그램 작성**

  % vi sample.l ⇒ 확장자는 .l

- **Lex 처리**

  % lex sample.l ⇒ lex.yy.c가 생김

- **컴파일**

  % cc lex.yy.c -ll ⇒ a.out이 생김

- **실행**

  % a.out ⇐ 키보드에서 입력

  % a.out < filename ⇐ 파일에서 입력

Lex의 입력

❑ 입력의 구성

정의부분
%%
규칙부분
%%
사용자 부프로그램 부분

- 정의부분
  - lex.yy.c에서 사용할 자료구조, 변수, 상수를 정의
  - 특정한 정규표현을 하나의 이름으로 정의 (규칙부분의 정규표현에서 {}속에 사용)
- 규칙부분
  - 토큰의 형태를 정규표현으로 나타냄
  - 토큰이 인식되었을때 처리할 행위를 기술
- 사용자 부프로그램 부분
  - 사용자가 작성한 프로그램을 lex.yy.c에 복사

정의부분

❑ 형태

```
%{
    /* lex.yy.c에 복사될 내용 */
%}
이름1     치환식1
이름2     치환식2
...
이름n     치환식n
```

➢ 정의 예

```
%{
    int count = 0;
%}
word                [^ \t\n]+
eol                 \n
```

규칙부분

❑ 형태

```
%%
정규표현1        행동1
정규표현2        행동2
...
정규표현n        행동n
```

- ❖ 행동: C 프로그램 또는 특수행동
- ❖ 특수행동: ECHO = 인식된 입력을 그대로 출력

규칙부분

❑ 정의 예

```
%%
{word}      count = count + 1;
{eol}       {count = count + 1; printf("%d", count);}
.            ECHO;
```

Lex 입력의 예

- **Lex** 입력의 가장 간단한 예

```
%%
.               ECHO;
```

- 숫자를 인식

```
%%
[0-9]*          printf("number");
.               ;
```

스트링 인식

- **규칙 1: 매치되는 정규표현이 없으면,**
  - default로 ECHO 행동을 수행
- **규칙 2: 매치되는 정규표현이 여러개 있으면,**
  - 길이가 긴 것을 우선적으로 매치
- **규칙 3: 길이가 같은 정규표현이 여러개 매치되면,**
  - 순서적으로 앞에 나오는 정규표현을 매치

Lex 이용하기

Lex 이용 (1)

(1) 숫자(정수와 실수)를 인식 (지수부분은 제외) [앞의 강의노트 참고]

(2) [1에 추가] 문자는 그대로 통과

(3) [2를 수정] 숫자를 출력

- `ECHO` 이용 (`printf("%s", yytext);`와 동일)

(4) [3을 수정] "인식=>*숫자*" 형태로 출력

- `printf()`문과 `yytext`를 이용
- hint: `printf("number => %s\n", yytext);`

(5) [4를 수정] 인식된 숫자의 길이도 함께 출력

- "인식=>*숫자*(*길이*)" 형태
- `yyleng` 이용
- hint: `printf("number => %s(%d)\n", yytext, yyleng);`

Lex 이용 (2)

(6) [5에 추가] 끝난후 끝났음을 출력

- 사용자 부프로그램 영역에 `main()` 추가
- `yylex()`를 호출

(7) word counting

- wc 명령어의를 구현

```
main() {
  yylex();
printf("end\n");
}
```

(8) if-문의 수 헤아리기

- 입력: C 프로그램
- 출력: if-문의 수
- hint: “int gif;”와 같이 “if”를 포함하는 식별자는 제외

(9) C에서 스트링 상수 인식

- 따옴표(“) 사이의 스트링
- 스트링에 escape 문자(\n, \”, \\ 등)가 포함될 수 있음

C 프로그램 분석하기

- **IF 문, WHILE 문, FOR 문의 갯수 헤아리기**
- **배정문(assignment statement)의 갯수 헤아리기**
  - 배정문 기호: =, +=, -=, ..., <<=, >>=
  - 관계연산자: ==, !=, <=, >=
- **주석(comment) 제거하기**
  - 주석이 한 줄내에서만 있을때
  - 주석이 두 줄이상 걸쳐서 있을때
- **주석제거후 테스트 방법**

```
a.out < inputfile > outputfile
diff inputfile outputfile | more
```

C 프로그램에서 IF 문의 갯수

```
%{
  int c=0;
%}
word       [a-zA-Z]+
%%
if         c++;
{word}     ;
.          ;
\n         ;
%%
main() {
  yylex();
  printf("number of IF statements => %d\n", c);
}
```

배정문의 갯수 헤아리기

```
%{
   int c=0;
%}
rop          (==|!=|<=|>=)
ass          (<<=|>>=)
%%
{ass}        c++;
{rop}        ;
=            c++;
.            ;
\n           ;
%%
main() {
   yylex();
   printf("number of assignment statements => %d\n", c);
}
```

주석 제거하기 (1)

- 주석이 한 줄내에서만 있을 때 (그리고 한줄에 한개만 있을 때)

```
%%
"/*".*"*/"      ;
.                       ECHO;
```

- 주석이 두 줄에 걸쳐서 있을 때 (또는 한줄에 여러개가 있을 때)
  - ???

상태기억자

❑ 특수한 상황하에서만 토큰을 인식

- ❖ 예: "aaa"가 먼저 나타난 후 "bbb"를 인식

❑ 사용법

- ❖ 정의부분

  %s NAME

- ❖ 규칙부분

  [a-z]　　BEGIN NAME;

  <NAME>[0-9]　　;

❑ 사용 예

```
%s AAA
%%
aaa          BEGIN AAA;
<AAA>bbb     BEGIN 0;
```

주석 제거하기 (2)

- 주석이 두 줄에 걸쳐서 있을 때

```
%s CMT
%%
"/*"            BEGIN CMT;
<CMT>"*/"       BEGIN 0;
<CMT>.                ;
<CMT>\n         ;
.                     ECHO;
```

Programming HW#1

❑ 문제 1

- ❖ 여는 괄호 "("와 닫는 괄호 ")"사이에 있는 문자 "a"의 갯수 헤아리기
- ❖ 예
  - ➢ 입력: `(abcda)abc(cba)`
  - ➢ 출력: `3`

❑ 문제 2

- ❖ 중괄호({}) 사이의 /*...*/는 주석으로 인식
- ❖ 그렇지 않은 /*...*/는 출력함
- ❖ 예
  - ➢ 입력: `abc{def/*ghi*/jkl}mno/*qrs*/tuv`
  - ➢ 출력: `abc{defjkl}mno/*qrs*/tuv`

Lex의 응용

❑ Lex로 가능한 일들

- ❖ 단순한 숫자 계산
  - ➢ 입력되는 숫자의 합을 구한다.
- ❖ 스트링 변환
  - ➢ 특정 스트링을 다른 스트링으로 바꾼다.
- ❖ HTML 생성
  - ➢ 입력으로부터 HTML 문서를 생성한다.
- ❖ HTML 태그 인식
  - ➢ HTML 문서에서 태그를 인식하여 출력한다.

어휘분석기의 설계

어휘분석기 개요

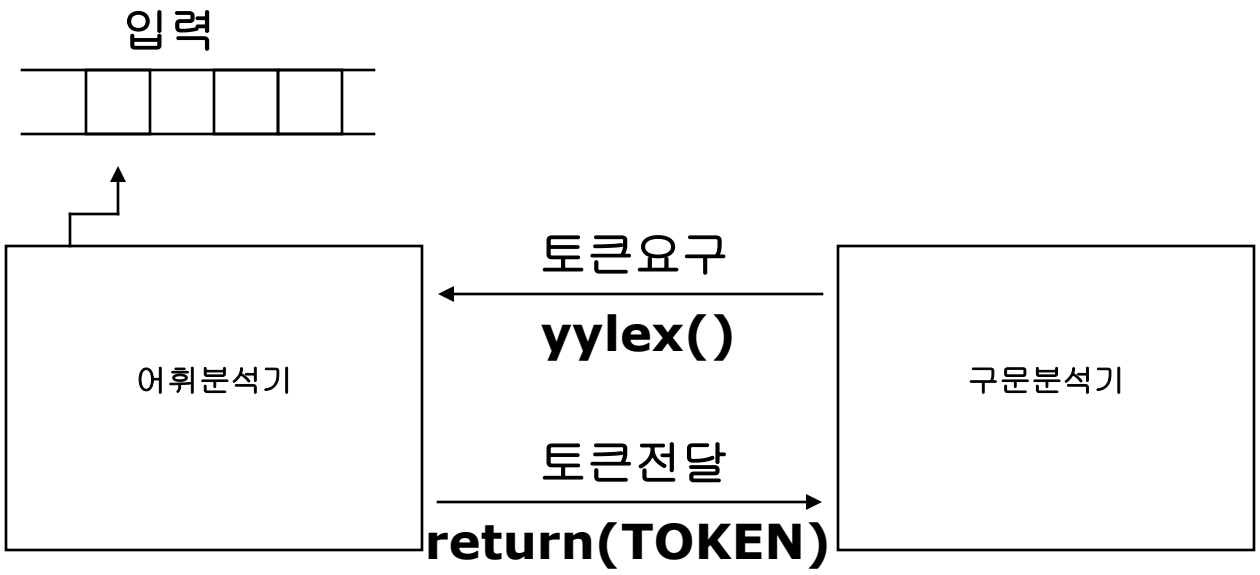
❑ 컴파일러의 단계

- ❖ 어휘분석의 결과 → 구문분석의 입력
- ❖ 구문분석기가 토큰을 요구, 어휘분석기가 토큰을 전달

❑ 어휘분석기의 역할

- ❖ 토큰인식 후 구문분석기에 전달
- ❖ 심볼테이블 구성

어휘분석기와 구문분석기



- 토큰 인식 후 yylex()의 실행을 종료
- 차후에, 구문분석기가 yylex()를 다시 호출

ToyPL의 어휘 구조

- 키워드
  - program, main, proc, func, returns, var, int, long, if, then, else, while, for, to, call, return, begin, end
- 연산자 및 구분자
  - 연산자: = & | ! < <= > >= != + - * /
  - 구분자: ; : ( ) , .
- 이름과 숫자
  - 이름: 영문자로 시작, 영문자와 숫자가 이어짐
  - 숫자: 정수의 10진수 표기
- 주석
  - /*와 */ 사이의 모든 문자열

토큰 설계 (1) - 키워드

- 각 키워드에 해당하는 토큰을 생성

program	⇒ TPROGRAM	main	⇒ TMAIN
proc	⇒ TPROC	func	⇒ TFUNC
returns	⇒ TRETURNS	var	⇒ TVAR
int	⇒ TINT	long	⇒ TLONG
if	⇒ TIF	then	⇒ TTHEN
else	⇒ TELSE	while	⇒ TWHILE
for	⇒ TFOR	to	⇒ TTO
call	⇒ TCALL	return	⇒ TRETURN
begin	⇒ TBEGIN	end	⇒ TEND

토큰 설계 (2) - 연산자

- 연산자를 나타내는 토큰을 생성

  = ⇒ TASS

  & ⇒ TAND | ⇒ TOR ! ⇒ TNOT

  < ⇒ TLT <= ⇒ TLE

  > ⇒ TGT >= ⇒ TGE != ⇒ TNE

  + ⇒ TPLUS - ⇒ TMINUS

  * ⇒ TMUL / ⇒ TDIV

- 구분자는 그 자체를 토큰으로 함

  ; : ( ) , .

토큰 설계 (3) - 식별자

- 식별자
  - 영문자로 시작, 영문자와 숫자가 0번 이상 계속
  - 토큰은 TWORD
  - 심볼테이블에서 항목을 검색, 없으면 항목 추가
  - 검색이나 추가 후, 테이블 인덱스를 함께 전달
- 숫자 리터럴
  - 숫자: 정수의 10진수 표기
  - 토큰은 TNUMBER
  - 숫자의 값을 함께 전달
- 주석
  - 어휘분석기가 제거함

심볼테이블

❑ 심볼테이블의 구조

```
typedef enum {PNAME,PROC, FUNC, INT, LONG} typeKind;
struct symbolTable {
        char     name[];
        typeKind type;
        int      init;
        int      addr;
} symtab[];
```

	이름	타입	초기값	주소
1	**fact**	**PNAME**		
2	**i**	**INT**		**5**
3	**sum**	**LONG**	**0**	**6**

Programming HW #2

- **ToyPL 어휘분석기 구현**

- **입력: ToyPL 프로그램**
- **출력:**
  - 인식된 어휘에 대한 토큰 이름을 출력한다.
  - 토큰이름은 강의노트에 정의되어 있음.
  - 토큰출력은 main() 함수에서 한다.

ToyPL 입력프로그램

```
/*
   input program
*/
program Sample
proc Fact (n : long)
  var m : integer;
  begin
    m = n - 1;
    call Fact(m);
  end;
/* main body */
main
var a, b : integer;
begin
  a = 0;
  call Fact(a);
end .
```