Lex

Joeun Park

April 18th, 2019

제 1 절 Lex

1.1 About Lex

Lex는 "scanner" 또는 "lexer"와 같은 어휘 분석기를 생성하는 컴퓨터 프로그램입니다.
Lex는 일반적으로 yacc parser generator와 함께 사용됩니다. 기존에 Mike Lesk와 Eric Schmidt에 의해 작성되었고, 1975년에 기술되었던 Lex는 많은 유닉스 시스템에서 사용되는 표준 어휘 분석기이며, 이에 상응하는 도구는 POSIX 표준의 일부로 명시 되어 있습니다.

Lex는 어휘 분석기를 지정하는 입력 스트림을 읽고, C 프로그래밍 언어로 lexer를 구현하는 소스 코드를 출력 합니다.
일정한 구조를 가진 입력을 받아들이는 프로그램에서 지속적으로 반복하는 작업은 입력 내용을 의미 있는 단위(unit)로 분해하여 그들 사이의 관련성을 파악하는 일입니다. 입력을 토큰이라고 하는 단위로 나누는 작업을 대개 어휘 분석(lexical analysis) 혹은 줄여서 렉싱(lexing)이라고 합니다.

1.2 Strucure of a Lex file

Lex 파일의 구조는 의도적으로 yacc 파일의 구조와 유사합니다. 파일은 항목에서 확인할 수 있는 바와 같이 퍼센트 기호만 포함하는 두 개의 줄로 구분된 세 부분으로 나뉩니다.

- definition section은 매크로를 정의하고 C로 작성된 헤더 파일을 가져옵니다. 여기에 C 코드를 작성하여 생성된 소스 파일에 그대로 복사 할 수도 있습니다.

- rules section은 정규 표현식 패턴을 C 구문과 연관 시킵니다. lexer가 지정된 패턴과 일치하는 input에서 텍스트를 봤을 때, 관련있는 C 코드를 실행합니다.

- user subroutine section에는 생성된 소스파일에 그대로 복사되는 C 명령문 및 함수가 들어 있습니다. 이 문장에는 rules section에 있는 규칙에 의해 호출된 코드가 포함되어 있을 것입니다. 대형 프로그램에서는 이 코드를 컴파일 할 때 별도의 파일에 배치하는 것이 더 편리합니다.

제 2 절 Problem Analysis

1. word는 1개 이상의 연속된 알파벳으로 이루어져 있어야 합니다. 다시 말해, 알파벳과 알파벳 사이에 mark로 설정한 글자가 끼어 들어간다면 이는 연속된 알파벳으로 이루어져 있지 않게 되므로 하나의 word라고 볼 수 없게 됩니다.

2. "<<"와 같이 "<"이 연속하여 이루어진 문장부호의 경우, "<"을 각 한 개로 카운트해야 합니다. 따라서 "<<"는 카운트 +1이 두 번 실행됩니다.

3. 출력 내용은 word, '=', '{', '}', mark, 그리고 'number' 각각 총 6개 항목의 글자 갯수입니다.

4. number는 정수형과 실수형 두 가지 타입을 가리킵니다. 단, 양수와 음수를 따로 구분하지는 않으며 소숫점을 갖는 숫자 까지만 number로 카운팅 하겠습니다.

5. 숫자와 숫자 사이에 있는 '.' 은 소숫점의 역할이므로 mark로 인식되지 않는 반면 이 밖의 경우에서 발견되는 '.' 은 mark로 인식 하겠습니다.

6. mark로 카운트 할 기호들은 다음과 같습니다.

```
".”|” <”|” >”|” ‘”|”~”|”!”|”@”|”#”|”$”|
”%”|”^”|”&”|”*”|”(”|”)”|” −”|”_”|”=”|
”+”|” ’”|”;”|”:”|” ,”|”/”|”?”|”\\”|”\””
```

backslash와 doublequot는 Lex에서 텍스트 그대로 인식하지 못하기 때문에 해당 텍스트 바로 앞에 또 다시 backslash를 붙여주었습니다.

제 3 절 Lex Code Description

3.1 Definition section

```
%{
        #include<stdio.h>
        int count_word = 0;
        int count_equal = 0;
        int count_Lbracket = 0;
        int count_Rbracket = 0;
        int count_mark = 0;
        int count_number = 0;
%}
```

definition section은 최종 프로그램에 포함하고자 하는 C 프로그램의 내용을 삽입하는 기능을 담당합니다. 삽입되는 내용은 출력을 위해 필요한 헤더 파일 및 분석된 코드의 카운트를 담당하는 변수입니다. 이 변수들은 그 다음의 rules section에서 사용됩니다. 이 때 해당 패턴이 나올 경우 그 값을 1씩 증가 시키며, 최종적으로 각 token의 갯수를 갖게 됩니다.

3.2 Rules section

```
%%
        [a-zA-Z]+                       {count_word++;}
        "="                             {count_equal++;}
        "{"                             {count_Lbracket++;}
        "}"                             {count_Rbracket++;}
        "."|"<"|">"|"‘"|"~"|"!"|"@"|"#"|"$"|"%"|"^"|"&"|
        "*"|"("|")"|"-"|"_"|"="|"+"|"'"|";"|":"|",|"/"|
        "?"|"\\"|"\""                   {count_mark++;}
        [\n\t ] ;
        [0-9]+                          {count_number++;}
        [0-9]+"."[0-9]+                 {count_number++;}
%%
```

rules section은 입력된 문자에서 매칭되는 문자열의 패턴 및 해당 패턴이 나타났을 때 수행되는 동작으로 구성됩니다.

- ([a-zA-Z])+
  a부터 z까지 혹은 A부터 Z까지의 문자로만 구성되는 문자열을 발견하면 변수 count_word의 값을 1 증가시킵니다.
- "="
  기호 '='를 발견하면 변수 count_equal의 값을 1 증가시킵니다.
- "{"
  기호 '{'를 발견하면 변수 count_Lbracket의 값을 1 증가시킵니다.
- 공백 및 줄바꿈은 고려 대상이 아닙니다.
- ([0-9])+
  0부터 9까지의 숫자로 구성되는 정수 형태의 수를 발견하면 변수 count_number의 변수값을 1 증가시킵니다.
- ([0-9]+"."[0-9]+)
  0부터 9까지의 숫자와 소숫점으로 구성되는 실수 형태의 수를 발견하면 마찬가지로 변수 count_number의 변수값을 1 증가시킵니다.

- Lex 코드에 정의되어 있는 특수문자를 발견하면 모두 mark로 간주하여 카운트 합니다.

3.3 User subroutine section

```
int main()
{
        yylex();
        printf("word = %d\n",count_word);
        printf("'=' = %d\n",count_equal);
        printf("'{' = %d\n",count_Lbracket);
        printf("'}' = %d\n",count_Rbracket);
        printf("mark = %d\n",count_mark);
        printf("number = %d\n",count_number);
        return 0;
}

int yywrap()
{
        return 1;
}
```

- main()
  각 token의 갯수를 출력 합니다.
  - yylex()
    Lex function에 해당되는 함수로써, 입력파일의 분석을 시작합니다.
  - printf()
    definition section에서 선언되었던 변수들의 값을 출력하는 함수입니다. 이 출력을 위해 definition section은 각각의 변수들과 헤더파일을 포함하고 있습니다.
- yywrap()
  yylex()와 마찬가지로 Lex function 중 하나로써, 파일의 EOF가 나타나면 호출되는 함수입니다. 이 함수가 1을 return하면 파싱의 종료를 지시합니다.

제 4 절 Execution of Lex

4.1 Test file

```
PI = 3.141592
"I love you."
fruit = {apple, strawberry, watermelon}
$1 = \1133.50
```

4.2 Result

```
word = 8
'=' = 3
'{' = 1
'}' = 1
mark = 7
number = 3
```

4.3 Analysis of the result

- WORD
  - PI
  - I
  - love
  - you
  - fruit
  - apple
  - strawberry
  - watermelon
- '='
  Equal sign is used 3 times.
- '{'
  Left curly bracket is used once.

- '}'
  Right curly bracket is also used once.
- MARK
  - Double quot(") is used twice.
  - Dollar sign($) is used once.
  - Backslash (which means Korean dollar) is used once.
  - Comma(,) is used twice.
  - Period(.) is used once.
- NUMBER
  - 3.141592
  - 1
  - 1133.50