Assignment 07: Lex

Date: 2024. 10. 18
Student ID:
Name: 김태훈

1. Lex의 인터페이스와 동작 구조[1]

Lex는 어휘 분석기를 자동으로 생성해주는 어휘 분석기 생성기이다. Lex는 사용자가 정의한 토큰을 정의한 정규 표현과 수행 코드를 입력받아 C언어로 작성된 어휘 분석기(lex.yy.c)를 출력한다. Lex의 입력 파일 형식은 다음과 같다.

```
declaration
%%
translation rules
%%
auxiliary procedures
```

각 부분은 %%로 구분된다. declaration 부분은 이름과 표현식으로 구성되며, 이름이 식별자, 표현식은 그 이름에 해당하는 정규 표현식이다. 예를 들면 다음과 같다.

```
Letter [a-zA-Z]
Digit [0-9]
```

이러한 방식으로 정규표현을 정의하는 것을 정규 정의(regular definition)이라 한다.

declaration 부분에는 변수, 상수, 매크로 부분이 포함되며 이는 %{ 와 %} 사이에 포함된다. 이 부분은 lex.yy.c에 그대로 들어간다. 예를 들면 다음과 같다.

```
%{
#include <stdio.h>
int count=0;
%}
Letter [a-zA-Z]
Digit [0-9]
```

정규 표현에서 쓰이는 표현들은 아래 표1과 같은 것이 있다.

기호	의미
“ ”	“ ” 속에 있는 문자는 텍스트 문자로 취급된다.
\	백슬레시는 하나의 문자를 이스케이프시킨다.
[]	[]안에 있는 기호 중 하나임을 의미한다.
*	영 번 이상의 반복을 의미한다.
+	한 번 이상의 반복을 의미한다.
.	줄바꿈 문자를 제외한 모든 문자를 의미한다.
^	(첫문자로 사용될 때는)행의 시작을 의미한다.
^	뒤에나오는 문자를 제외한 것을 의미한다. [^+]는 +를

	제외한 문자를 의미한다.
$	행의 끝을 의미한다.

<표1: 정규 표현에 쓰이는 여러가지 기호>

translation rules에는 변환 규칙이 들어가며, 표현식(또는 declaration 부분에서 정의한 표현식의 이름)과 수행 코드로 구성된다. 이름은 {}로 감싼다. 예를 들면 다음과 같다.

```
{Digit} printf("found digit\n");
[a-zA-Z] count++;
```

auxiliary procedures에는 변환 규칙에 사용되는 부프로그램을 사용자가 추가적으로 정의한다. 새로운 함수를 정의할 수도 있고, Lex에서 사용하는 함수 내부의 내용을 약간 수정할 수 도 있다. 예를 들어 yywrap()은 Lex에서 입력의 끝을 만났을 때 호출하는 함수로, 정상적으로 종료되려면 1을 반환한다. 이를 약간 수정하여 끝을 만났을 때 끝났음을 알리는 문자열을 출력하려면 아래와 같이 할 수 있다.

```
int yywrap() {
        printf("yylex done!\n");
        return 1;
}
```

Lex에서 쓰이는 함수와 전역 변수는 아래와 같은 것이 있다.

종류	이름	의미
전역변수	yytext	정규 표현에 의해 실제로 매칭된 문자열을 갖는다.
	yyleng	매칭된 문자열의 길이를 저장한다.
함수	yylex()	Lex 파일에서 정의된 정규 표현과 일치하는 문자열을 찾는다.
	yywrap()	Lex가 입력의 끝을 만났을 때 호출된다.
	yymore()	현재 매칭된 문자열 끝에 다음에 인식될 문자열을 덧붙인다.
입출력 함수	input()	입력 문자열로부터 다음 문자열을 읽는다
	output(c)	출력 문자열로 문자를 내보낸다.
	unput(c)	문자를 입력 문자열로 되돌려보낸다.

Lex의 동작 구조는 다음과 같다. Lex 파일을 작성 후(*.l)

```
flex line.l
```

을 통해 lex.yy.c 파일로 바꾼다. 이를 gcc를 사용하여 컴파일하거나, 기타 정의된 다른 c 파일과 함께 다음과 같이 컴파일한다.

```
gcc lex.yy.c other.c -o test
```

이제 여기에 텍스트파일을 넣으면 결과가 출력된다.

```
test < data.txt

digit found!
digit found!
digit found!
```

Lex의 동작 흐름은 다음과 같다.

1. 입력 파일이나 문자열을 yylex()가 한 글자씩 읽어들이면서, 정규 표현이나 정규 정의로 정의된 패턴과 비교한다
2. 일치하는 패턴이 있으면 해당하는 액션 코드가 실행된다.
3. 액션이 완료되면 yylex()는 입력을 계속 읽어들인다.
4. 입력의 끝에 도달하면 yywrap() 함수가 호출되고, 종료 여부를 결정한다.

2. Lex 입력 코드와 설명

```
%{
#include <stdio.h>
int word_count = 0;
int symbol_count = 0;
%}
word [a-zA-Z]+
symbol [\!\@\#\$\%\^\&\*\(\)\-\_\+\=\|\\\/\<\>\:\;\"\'\~\{\}\[\]\.\,\`]
space [ ]
newline [\n]
%%
{word} { word_count++; }
{symbol} { symbol_count++; }
{space} {}
{newline} {printf("%d %d\n",word_count,symbol_count);}
%%
int yywrap() {
    return 1;
}
```

(1) declaration

printf를 사용하기 위해 stdio.h를 include 하고, 단어 개수와 심볼 개수를 카운트할 변수를 선언하고, 0으로 초기화한다.

(2) regular definition

정규식에 이름을 붙여 정의한다. 단어, 특수기호, 공백, 줄바꿈 정규식을 정의했으며 의미는 다음과 같다.

1) word [a-zA-Z]+

word는 알파벳으로만 이루어진 단어를 정의하며, 소문자(a-z), 대문자(A-Z) 가 여러 개 연속으로(+) 이루어진 것으로 정의한다.

2) symbol [\!\@\#\$\%\^\&*\(\)\-_\+\=\|\\\/\<\>\:\;\"\'\~\{\}\[\]\.\,\`]

symbol은 세어야할 특수기호를 정의하며, lex에서 쓰이는 특수기호와 구분하기 위해 앞에 \ 를 넣어서 정의한다. !,@,#,$,%,^,&,*, (,),-, _, +, =, |, ₩, /, <, >, :, ;, ", ', `, ~, {, }, [,], .에 추가로 쉼표(,) 를 나타낸다.

3) space [ ]

space는 공백을 정의하며 두 bracket 사이에 공백 하나를 넣어 정의한다.

4) newline [₩n]

newline은 줄바꿈을 정의하며 두 bracket 사이에 ₩n을 넣어 정의한다.

(3) translation rules

1) {word} { word_count++; }

단어를 만났을 때, 단어 개수를 1 증가시킨다.

2) {symbol} { symbol_count++; }

세어야할 특수기호를 만났을 때, 심볼 개수를 1 증가시킨다.

3) {newline} {printf("%d %d₩n",word_count,symbol_count);}

newline을 만났다는 것은 한 줄의 입력이 끝난 것으로, 세어진 word와 symbol 개수(word_count와 symbol_count)를 출력한다.

4) {space} {}

공백을 만났을 때는 아무것도 실행하지 않는다.

(4) auxiliary procedures

1) yywrap

yywrap 함수는 Lex에서 입력이 끝났을 때 호출되며, 여기서는 1을 반환해 프로그램이 종료되도록 한다.

3. Lex 대신 직접 프로그램을 작성하는 방법 스케치

<S> ::= <alpha>{<alpha>} | <mark>
<alpha> ::= “a”|“b”|“c”| … “Y”|“Z”
<mark> ::=“(”|“)”|“-”|“+”|“=”| … |“,”

일 때 2를 직접 프로그램으로 작성하면 다음과 같다.

아래 코드를 간단히 설명하면 yylex() 함수는 한 줄 문자를 입력받아 여러 상황에 대해 다음과 같이 처리한다.

구분	처리
공백을 만났을 때	건너뛴다(pos++)
문자(알파벳)를 만났을 때	해당 위치부터 문자(알파벳)가 나오지 않을 때 까지 오른쪽으로 이동한다. 그리고 단어 개수(word_count)를 1 증가시킨다.
특수기호를 만났을 때	오른쪽으로 이동 후, 특수기호 개수(symbol_count)를 1증가시킨다.
그외	건너뛴다.

이를 gcc를 사용하여 컴파일 후 실행하면 입력과 출력 예는 다음과 같다.

입력: hello’hello
출력: 2 1
입력: I’d like to talk about (compiler).
출력: 7 4

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define INPUT_LEN 100
char input[INPUT_LEN];
int pos =0;
int word_count =0;
int symbol_count =0;
int custom_gets()
{
    if(!fgets(input, sizeof(input), stdin))
{
        return 0;
}
    return strlen(input);
}
int yywrap()
{
    return 1;
}
void yylex()
{
    while(custom_gets() >1)
{
        pos =0;
        word_count=0;
        symbol_count=0;
        while(input[pos] !='\0')
{
            while(input[pos] ==' ')
                pos++;
            char ch =input[pos];
            if(isalpha(ch))
{
                while(isalpha(input[pos]))
{
                    pos++;
}
                word_count++;
}
            else if(issymbol(ch))
{
                pos++;
                symbol_count++;
}
            else{
                pos++;
}
}
}
        printf("%d %d\n", word_count, symbol_count);
}
    yywrap();
    return;
}
```

References

[1] 박두순 저. “컴파일러의 이해”. 한빛미디어. 2020