프로그래밍 언어 hw2

B711016 김길호

April 13, 2021

1 과제 1

주어진 tc파일에 love라는 단어는 첫 글자인 l이 대문자, 소문자인 경우만 존재했습니다. 이 둘을 매칭하기 위해서 문자열을 완전 매칭하는 문법을 찾아 본 결과 큰 따옴표로 묶어주면 묶여진 단어를 매칭할 수 있었습니다. 이에 규칙절에서 "love", "Love" 두 단어 자체를 매칭하여 해결할 수 있었습니다.

2 과제 2

정규 표현식으로 표현하기위해 주어진 패턴의 ~는 +로 바꿔주었고, 완전한 매칭을 위해 시작과 끝을 제한해주는 문법기호인 웃음 표(ctrl+6)와 달러(ctrl+4)를 각각 맨 앞과 맨 뒤에 붙혔습니다. 이에 규칙절에서 해당 패턴과 완전히 매칭될 때 카운트할 수 있었습니다.

3 과제 3

3.1 Lex란?

Lex란 작성된 코드를 문법적으로 의미있는 토큰 단위로 분해하여 해당 토큰을 분석할 수 있게 도와주는 어휘 분석기입니다. 토큰은 주로 정규 표현식의 형태로 정의되며, 그 위치는 아래에서 언급할 렉스의 세 구조중 두 번째에 위치한 규칙절에 선언하여 인식하길 원하는 문자들을 어떤 규칙에 따라 정의할지 정할 수 있습니다. Lex의 구조는 크게 세 가지로 두 개의 %%로 구분되어집니다.

3.1.1 정의절

정의절을 %{과 %}으로 구분된 리터럴 블록과 (규칙절에서 사용할)변수 선언등을 담습니다. 리터럴 블록 내부에서는 c코드로 정의문과 같은 내용을 담을 수 있고, 변수 선언을 할 수 있습니다. 끝은 %%로 구분됩니다.

3.1.2 규칙절

지정할 토큰 규칙을 정의하는 공간이며 정의절에서 정의한 변수와 정규 표현식을 이용하여 패턴을 단순화 할 수 있고, 정의한 패턴이 매칭되었을 때 {과 } 내부에 c코드를 작성하여 수행할 동작을 정의할 수 있습니다. 끝은 %%로 구분됩니다.

3.1.3 사용자 서브루틴절

yylex()함수와 정의절에서 정의한 변수를 출력하는 등의 사용자가 원하는 C 루틴으로 구성되는 부분입니다.

3.2 구현 내용 및 코드 설명

3.2.1 정의절 구현

리터럴 블록 내부에는 기본적인 c코드를 사용하기 위해 필요한 stdio헤더와 단어를 체크할 때 문자열과 관련된 함수를 사용하기에 string헤더를 선언하였습니다. 규칙에 매칭되는 토큰들의 개수를 세기 위한 변수들, 단어의 개수를 셀 때 p의 개수를 세기 위한 변수, e로 시작하고 마지막 글자가 m인 단어를 찾았을 때 체크할 플래그 변수들도 같이 선언해주었습니다.

리터럴 블록 외부부터 %%까지 공간에는 아래 규칙절에서 사용할 간단한 변수를 정규식으로 표현했습니다. 그 종류와 의미는 아래와 같습니다.

1. DIGIT [0-9] : 0∼9가 scan값으로 들어올 때 이를 변수 DiGIT로 정의합니다.
2. LETTER [a-zA-Z] : 대시로 이어진 알파벳값들 중 scan값으로 들어올 때 이를 변수 LETTER로 정의합니다.
3. OCTAL [0][0-7]+ : 8진수는 맨 앞에 0과 그 뒤에 0-7사이의 값으로 이루어져있기에 scan값이 들어오면 이를 변수 OCTAL로 정의합니다.
4. OPERATOR 연산자들··· : 문제에서 정의한 연산자들이 scan값으로 들어오면 이를 변수 OPERATOR로 정의합니다.
5. OPEN, CLOSED, EQUAL : 괄호나 대입연산자가 scan값으로 들어오면 각각 변수들로 정의합니다.
6. %x COMMENT : /*주석을 처리하기 위해 사용할 시작 상태로 /*주석을 만났을 때 <COMMENT>으로 분기합니다.

3.2.2 규칙절 구현

규칙 절의 먼저 작성한 규칙이 먼저 선택된다는 개념을 생각해봤을 때, 규칙들의 순서(우선 순위)가 원하는 문자열을 매칭하는 것에 크게 의미가 있다고 생각했습니다. 따라서 카운트 해야할 규칙들의 순서를 어떻게 구성했는지, 그 규칙들은 어떻게 이루어지는지를 서술하겠습니다.

순서

1. 주석문 : 주석문 안의 값은 어떤 것이라도 카운트하지 않으므로 주석문이 제일 먼저 선택돼야한다고 생각했습니다. 구현 방식에 대해서 설명하자면 아래와 같이 작성하였습니다.

"/*" {BEGIN(COMMENT); ++comment;} : 여는 주석을 만날 때 COMMENT로 분기합니다.
"//".*개행 { ++comment;} : 큰 따옴표와.* 정규 표현식을 사용해 주석과 문자들을 스캔해줬고, 마지막에 개행도 포함하였습니다.
<COMMENT>.|개행 ; : /*부터 나오는 모든 문자와 개행을 만날 때 마다 eat해줍니다.
<COMMENT>"*/" {BEGIN(INITIAL);} : 마지막으로 닫는 주석 */을 만났을 때, 첫 %% 아래인 초기 규칙절로 돌아갑니다. 물론 닫는 주석 이후의 개행은 포함하지않습니다.

2. 전처리문 : 주석문 다음으로 다른 모든 값을 무시할 수 있는 규칙이라고 생각하였고, 큰 따옴표와.* 정규 표현식을 사용해 아래와 같이 전처리문 문장의 문자,단어뿐만 아니라 개행까지 모두 읽어 전처리문으로 인식하게 작성하였습니다.

"#include".*개행 { preprocessor++;}
"#define".*개행 { preprocessor++;}

3. 단어 : 단어는 알파벳으로만 이루어지는 문자열을 매칭하였습니다. 매칭한 후 단어의 종류를 확인하기 위해서 문자열을 순회하며 p의 개수를 전역변수를 이용하여 세었고, 첫 문자가 e이며 마지막 문자가 m인지를 판단하여 전연변수 플래그를 체크하였습니다. 이에 wordcase1, wordcase2에 해당되지않는 단어는 그냥 단어의 개수를 올려주었고 마지막에 사용한 전역변수를 모두 0으로 초기화해주었습니다.

{LETTER}+ {
for (;i < strlen(yytext);i++) if (yytext[i] == 'p') countp++;
if (yytext[0] == 'e' && yytext[strlen(yytext)-1] == 'm') flag = 1;
if(countp == 2) {wordcase1++;}
else if (flag) {wordcase2++;}
else {word++;}
i = 0; countp = 0; flag = 0;
}

4. 10진법 음수, 10진법 양수, 8진법 숫자, 연산자 : 4개로 묶어준 이 규칙들은 넷이 서로 순서가 상관이 있기에 같이 묶어주었습니다. 우선 10진법 양수와 음수는 첫 부호만 제외하곤 같은 방식이기에 음수가 먼저 나와야했고, 10진법 음수의 -를 자칫 연산자로 처리해버릴 수 있기에 연산자를 10진법 음수 뒤에 두었습니다. 8진수는 어차피 0으로 시작하기에 우선 순위에 크게 상관이 없었습니다.아래는 그 구현 방식입니다.

·10진법 음수 : 음수의 경우 -로 시작하므로 -뒤에 바로 나올 수 있는 수의 범위를 [1-9]으로 거르고 {DIGIT}* 정규식으로 [0-9]수의 반복까지 매칭하였습니다.
"-"[1-9]{DIGIT}* {negative_number++;}

·8진법 숫자 : 정의절에서 선언한 변수를 사용하여 매칭하였습니다.
{OCTAL} {octal_number++;}

·연산자 : 정의절에서 선언한 변수를 사용하여 매칭하였습니다.

{OPERATOR} {operator++;}

·10진법 양수 : 양수의 경우 음수와 같은 방식으로 걸러 주었으나 부호가 없는 수들만 10진법 양수로 판정했습니다.
[1 − 9]DIGIT* {posititive_number++;}

4. 대입 연산자 : 대입 연산자는 앞서 정의한 ==연산자와 겹칠 수 있기에 연산자보다 뒤에 배치하였습니다.
{EQUAL} {equal++;}

5. count되지 않은 문자(mark) : 정규표현식 .|개행을 사용하여 위의 규칙절에서 매칭되지 않은 문자들을 매칭해주었습니다.
.|개행 {mark++;}

3.2.3 사용자 서브루틴절 구현

스캐너역할을 하는 yylex함수를 사용하였고, 이후 전역변수로 카운트한 변수들을 출력형식에 맞게 출력하였습니다.