프로그래밍 언어 HW2

B811181 조예린

April 7, 2021

1 과제 1

텍스트에서 'love'라는 단어가 몇 번 나오는지 카운트하여 출력하라.

1.1 정의절

정의절에서 love의 카운트 횟수를 저장할 int형 변수를 선언했다.

```
%{
#include <stdio.h>
int n_love = 0;
%}
```

1.2 규칙절

1. love 또는 Love를 인식한 경우 변수의 값을 증가시켰다.

```
love|Love                    {n_love++;}
```

2. love 또는 Love 가 아닌 문자나 줄바꿈 문자의 경우 아무것도 하지 않고 흘려주었다.

```
.|\n     ;
```

1.3 서브루틴절

love의 개수를 출력한다.

```
int main(){
        yylex();
        printf("number of love=%d\n", n_love);
        return 0;
}

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

2 과제 2

(100∼1∼ |01)∼을 만나면 'is danger'을 출력하라.

2.1 규칙절 : 패턴

```
(10(0+)1((1*)|0)1)+        {printf("%s is danger\n", yytext);}
```

규칙과 동일하게 패턴을 적용하였다.
다만 $'1 \sim |0'$ 부분의 구현을 살펴보면 무조건 앞에 1이 온 후 $(1*)|0$으로, 즉 1이 0번 이상 반복되거나 0이 무조건 한번 나와야하는 패턴을 짰다.
yytext로 읽어들인 문자 그대로 반환하여 'is danger'을 출력하였다.

3 과제 3

c코드를 읽어 해석한 것을 출력하라.

3.1 정의절

3.1.1

각 패턴의 개수를 저장할 배열을 선언하였다.

```
%{
#include <stdio.h>
int n_arr[13] = { 0, };
%}
```

3.1.2

규칙절에서 사용할 규칙을 정의하였다.

DIGIT	[0−9]
LETTER	[a–zA–Z]
NOTP	[a–oq–zA–OQ–Z]

3.2 규칙절

lex에서는 먼저 작성한 패턴이 우선순위가 높으므로 과제설명 파일에 기재된 순서와 상관없이 우선순위가 높은 순서로 보고서를 작성하겠다.

3.2.1 comment

: 주석문의 개수 (n_arr[5]에 저장)

(1) 여러줄 주석 (/*...*/)

```
"/*"(.*|\n*)*"*/"          {n_arr[5]++;}
```

우선 /*와 */는 문자 그대로 해석하기 위해 큰 따옴표("")로 감싸주었다.
"/*"와 "*/"로 감싸진 중간 패턴을 살펴보자.
주석 안에 있는 어떤 코드든 주석이 끝날 때까지 무의미하게 흘려보내야하기 때문에 '.'과 줄바꿈 문자를 사용하였다. 어떤 문자나 줄바꿈 문자가 있든 없든, 또는 둘 중 어떤 것이 오거나 반복되는지에 상관없이 다 인식할 수 있도록 이와 같은 패턴으로 작성하였다.

(2) 한줄 주석 (//...)

```
"//"(.)*\n          {n_arr[5]++;}
```

한줄 주석은 //으로 시작한 후 줄바꿈 문자가 나오면 끝나므로 위와 같이 작성하였다.

3.2.2 Preprocessor

: ♯include, ♯define 전처리문의 개수 (n_arr[0]에 저장)

```
(#include|#define)(.*)\n          {n_arr[0]++;}
```

♯include 혹은 ♯define이 나오면 그 한 줄을 전처리문으로 인식한다.

3.2.3 octal number

: 8진법 숫자 개수 (n_arr[1]에 저장)

```
0{DIGIT}+          {n_arr[1]++;}
```

c언어에서 8진수는 0으로 시작하는 숫자이기 때문에 앞에 0이 나오고 그 뒤에 숫자가 1번 이상 나와야하는 패턴으로 작성하였다. (0이 한번만 나온다면 숫자 0을 의미하므로 1번 이상 반복으로 해주었다.)
이 때 정의절에서 미리 정의해둔 DIGIT을 사용하였다.

3.2.4 negative decimal number

: 10진법 숫자 중 음수의 개수 (n_arr[2]에 저장)

```
-{DIGIT}+          {n_arr[2]++;}
```

십진법 음수는 '-'로 시작하고 숫자가 한개 이상 있어야하므로 이와 같은 패턴으로 구현하였다.

3.2.5 positive decimal number

: 10진법 숫자 중 양수의 개수 (n_arr[3]에 저장)

```
{DIGIT}+            {n_arr[3]++;}
```

0으로 시작하는 숫자, 즉 8진법은 위에서 인식됐으므로 이와 같이 코드를 작성해도 가능하다.

3.2.6 operator

: 연산자의 개수 (n_arr[4]에 저장)

```
"+"|"-"|"*"|"/"|"%"          {n_arr[4]++;}
==|!=|>|<|>=|<=              {n_arr[4]++;}
&&|"||"|!                    {n_arr[4]++;}
"++"|"--"                    {n_arr[4]++;}
","|"&"|"*"|"->"             {n_arr[4]++;}
```

3.2.7 '=' '{' '}'

: 각 기호의 개수 (n_arr[6],[7],[8]에 각각 저장)

```
=           {n_arr[6]++;}
"{"         {n_arr[7]++;}
"}"         {n_arr[8]++;}
```

3.2.8 wordcase

(1) wordcase1 : p가 두 번만 들어간 단어의 개수 (n_arr[9]에 저장)

```
{NOTP}*p{NOTP}*p{NOTP}*  {n_arr[9]++;}
```

정의절에서 미리 구현해둔 NOTP를 사용하였다.
p가 정확히 두 번 존재해야하기 때문에 pp의 앞, 중간 사이, 끝에 NOTP*를 넣어 주었다.

(2) wordcase2 : e로 시작하고 마지막 글자가 m인 단어의 개수 (n_arr[10]에 저장)

```
e{LETTER}*m         {n_arr[10]++;}
```

정의절에서 미리 구현해둔 LETTER를 사용하였다.

3.2.9 word

: 그 외 단어의 개수 (n_arr[11]에 저장)

```
{LETTER}+           {n_arr[11]++;}
```

3.2.10 mark

: 위에서 count 되지 않은 문자의 개수 (n_arr[12]에 저장)

```
.|\n      {n_arr[12]++;}
```

3.3 서브루틴절

정해진 형식에 맞춰 결과를 출력한다.

```
int main(){
        yylex();
        printf("preprocessor = %d\n", n_arr[0]);
        printf("octal number = %d\n", n_arr[1]);
        printf("negative decimal number = %d\n", n_arr[2]);
        printf("positive decimal number = %d\n", n_arr[3]);
        printf("operator = %d\n", n_arr[4]);
        printf("comment = %d\n", n_arr[5]);
        printf("'=' = %d\n", n_arr[6]);
        printf("'{' = %d\n", n_arr[7]);
                printf("'}' = %d\n", n_arr[8]);
        printf("wordcase1 = %d\n", n_arr[9]);
        printf("wordcase2 = %d\n", n_arr[10]);
        printf("word = %d\n", n_arr[11]);
        printf("mark = %d\n", n_arr[12]);
        return 0;
}

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