project1. scanner

2015004302 CSE 곽상원

목차

---

1. Compile 환경

- Ubuntu14.04, gcc, flex
- “make all” command를 이용해 cminus, cminus_flex를 동시에 빌드 및 생성

2. Project purpose

- 기존의 Tiny를 위한 Scanner source file을 일부 변경해서 Cminus를 위한 Scanner를 만든다.
- Cminus Scanner를 위해 필요한 토큰과 그에 대한 처리방법을 추가한다.
- **❶scan.c**나 **❷lex input파일(cminus.l)**을 수정해서 주어진 입력을 처리할 수 있는 Scanner를 만든다.

< 입력(토큰) table >

keyword	else(ELSE), if(IF), int(INT), return(IF), void(VOID), while(WHILE)
Symbol	“+”(PLUS), “-”(MINUS), “*”(TIMES), “/”(OVER), “<”(LT), “<=”(LE), “>”(GT), “>=”(GE), “==”(EQ), “!=”(NE), “=”(ASSIGN), “;”(SEMI), “,”(COMMA), “(”(LPAREN), “)”(RPAREN), “[”(LBRACE), “]”(RBRACE), “{”(LCURLY), “}”(RCURLY), “/*”(LCOM), “*/”(RCOM)
그 외	letter+(ID), digit+(NUM), “\n”, whitespace, EOF(ENDFILE), .(ERROR),

▶source file 수정사항

globals.h: 위와 같이 새로운 토큰을 추가했다

scan.c: 입력에 대해 새로운 토큰을 처리하기 위해 state 및 case를 추가했다.

util.c: 새로운 토큰에 대한 출력형태를 추가했다.

cminus.l: 위의 입력을 적절히 처리할 수 있도록 rule부분에 내용을 추가했다.

3.프로그램 흐름

- input stream을 읽어들어 입력에 적절한 토큰을 매칭하고 반환한다.
- 반환된 토큰에 대해 정해진 형태의 출력을 진행한다.
- 이를 입력으로 EOF가 올 때까지 반복한다.

4.각 토큰에 대한 처리

▷ 토큰 획득 방법

❶ scan.c를 수정한 경우

▶ 하나의 입력으로 토큰을 결정할 수 있는 경우

⑴ EOF(ENDFILE), “,”(COMMA), “+”(PLUS), “[”(LBRACE), “]”(RBRACE), “{”(LCURLY), “}”(RCURLY), “-”(MINUS), “*”(TIMES), “(”(LPAREN), “)”(RPAREN), “;”(SEMI), .(ERROR)

START state에서 시작해서 곧 바로 DONE으로 가고, 입력에 대한 적절한 토큰을 반환한다.

▶그 외의 경우

⑴ letter+(ID), “else”(ELSE), “if”(IF), “int”(INT), “return”(RETURN), “void”(VOID), “while”(WHILE)

문자를 하나 입력받으면 INID state로 이동한다. 그 후 문자이외의 것을 입력받으면 DONE로 가고 ungetNextChar()를 호출해 방금 읽어 들인 입력을 다음번에 읽을 수 있도록 한다. 그리고는 버퍼에 저장된 입력에 대해 keyword검사를 한다. keyword가 맞다면 적절한 keyword 토큰을 아니면 ID를 반환한다.

⑵ digit+(NUM)

숫자를 하나 입력받으면 INNUM state로 이동한다. 그 후 숫자이외의 것을 입력받으면 DONE으로 가고 ungetNextChar()를 호출한다. 끝으로 NUM을 반환한다.

⑶ **whitespace(공백), “\n”(줄바꿈)**

해당 입력을 저장하지 않고 START state에 머문다. 결국 다른 입력을 받을 때까지 루프를 돌게 된다.

⑷ **“=”(ASSIGN), “==”(EQ)**

‘=’를 입력받으면 INEQ state로 이동한다. 그 후의 입력이 ‘=’이면 DONE으로 이동 후 EQ를 반환한다. 아닌 경우, ungetNextChar()호출 후 DONE으로 이동하고 ASSIGN을 반환한다.

⑸ **“<”(LT), “<=”(LE)**

‘<’를 입력받으면 INLT state로 이동한다. 그 후의 입력이 ‘=’이면 DONE으로 이동 후 LE를 반환한다. 아닌 경우, ungetNextChar()호출 후 DONE으로 이동하고 LT를 반환한다.

⑹ **“>”(GT), “>=”(GE)**

‘>’를 입력받으면 INGT state로 이동한다. 그 후의 입력이 ‘=’이면 DONE으로 이동 후 GE를 반환한다. 아닌 경우, ungetNextChar()호출 후 DONE으로 이동하고 GT를 반환한다.

⑺ **“!=”(NE)**

‘!’를 입력받으면 INNE state로 이동한다. 그 후의 입력이 ‘=’이면 DONE으로 이동 후 NE를 반환한다. 아닌 경우, ungetNextChar()호출 후 DONE으로 이동하고 ERROR를 반환한다.

⑻ **“/”(OVER), “/*”(LCOM)**

‘/’를 입력받으면 INOVER state로 이동한다. 그 후의 입력이 ‘*’이 아닌 경우, DONE으로 이동 후 ungetNextChar()호출하고 OVER를 반환한다. 그 후의 입력이 ‘*’인 경우, INCOMMENT로 이동한다. 이때, 버퍼에 “/*”는 모두 저장되지 않는다.

INCOMMENT state는 주석 안에 있음을 나타내는 state로, 버퍼에 입력이 저장되지 않는다. 이 state에서 ‘*’을 만나면 INCOMMENT_로 이동한다. 아닌 경우 INCOMMENT에 머문다. (예외사항으로, EOF를 만나는 경우는 ENDFILE을 반환한다.)

INCOMMENT_에서 ‘/’을 입력받으면 START state로 이동한다. 아닌 경우 INCOMMENT state로 돌아간다.(예외사항으로, EOF를 만나는 경우는 ENDFILE을 반환한다.)

❷ lex input파일(cminus.l)을 수정한 경우

⑴ **“/*”을 입력으로 받는 경우**

<code>

```
char cur, next;
```

```
cur=input();
while(1){
        f(cur==EOF) break;
        else{
        next=input();
         if(cur=='\n') lineno++;
        if((cur=='*')&&(next=='/')) break;
        cur=next;
        }
}
```

바로 다음 입력을 cur에 저장한다. 만약 cur이 EOF인 경우, 즉시 while문을 탈출한다. 아닌 경우에 대해, 다음 입력을 next에 저장한다. 그리고,

1) cur이 '\n'인 경우, 줄 번호를 가리키는 lineno를 1증가 시킨다.
2) cur이 '*'이고 next가 '/'이면 처음 입력 "/*"과 쌍을 이루어 주석을 벗어나게 되므로 while문을 탈출한다.
3) 둘 다 아닌 경우, cur에 next값을 넣고(이런 식으로 계속 다음 입력을 받아들임) while문 처음으로 돌아간다.

⑵ 그 외의 경우

입력에 대한 적절한 토큰을 반환한다.

ex) "<" {return LT;}

▷토큰에 대한 출력

- 입력받은 각 토큰에 대해 util.c에 정의해놓은 형태로 모니터에 출력한다.

ex) case LT: fprintf(listing,"<\n");//listing은 stdout

5. 실행예제 및 결과

▶입력

```
1 int main(void){
2     int i=0;/*number*/
3     char arr[3]={0,1,2};
4     while(arr[i++] > 1) return;
5 }/*easy program*
```

▶실행결과

❶ scan.c를 수정한 경우

```
ksw@ubuntu:~/Downloads/Compiler_project/temp$ ./cminus test.c

C-MINUS COMPILATION: test.c
   1: int main(void){
        1: reserved word: int
        1: ID, name= main
        1: (
        1: reserved word: void
        1: )
        1: {
   2:   int i=0;/*number*/
        2: reserved word: int
        2: ID, name= i
        2: =
        2: NUM, val= 0
        2: ;
   3:   char arr[3]={0,1,2};
        3: ID, name= char
        3: ID, name= arr
        3: [
        3: NUM, val= 3
        3: ]
        3: =
        3: {
        3: NUM, val= 0
        3: ,
        3: NUM, val= 1
        3: ,
        3: NUM, val= 2
        3: }
        3: ;
```

```
4: reserved word: while
4: (
4: ID, name= arr
4: [
4: ID, name= i
4: +
4: +
4: ]
4: >
4: NUM, val= 1
4: )
4: reserved word: return
4: ;
5: }
6: EOF
```

❷ lex input파일(cminus.l)을 수정한 경우

```
ksw@ubuntu:~/Downloads/Compiler_project/temp$ ./cminus_flex test.c

C-MINUS COMPILATION: test.c
        1: reserved word: int
        1: ID, name= main
        1: (
        1: reserved word: void
        1: )
        1: {
        2: reserved word: int
        2: ID, name= i
        2: =
        2: NUM, val= 0
        2: ;
        3: ID, name= char
        3: ID, name= arr
        3: [
        3: NUM, val= 3
        3: ]
        3: =
        3: {
        3: NUM, val= 0
        3: ,
        3: NUM, val= 1
        3: ,
        3: NUM, val= 2
        3: }
        3: ;
```

```
4: reserved word: while
4: (
4: ID, name= arr
4: [
4: ID, name= i
4: +
4: +
4: ]
4: >
4: NUM, val= 1
4: )
4: reserved word: return
4: ;
5: }
6: EOF
```