Compiler Scanner Report

2019056799 차다윤

1. Compilation method and environment

Ubuntu 환경에서 실행한다.

Method1) 주어진 makefile로 Make 명령어를 실행하면 cminus_cimpl 실행파일이 생성되는데, ./cminus_cimpl [lexical analysis를 수행할 파일 이름] 을 실행하면 Scanner의 실행 결과가 출력된다.

Method2) Make 명령어를 실행하면 cminus_lex 실행파일이 생성되고, ./cminus_lex [파일 이름] 을 실행하면 Scanner의 실행 결과가 출력된다.

2. Explanation about how to implement and how to operate

Method1) globals.h에서 tiny 의 토큰들을 C-Minus의 토큰들로 변경했다.

Util.c에서 printToken 함수에 추가된 토큰들에 대한 case들을 추가했다.

Scan.c에서 reservedWords 구조체의 단어들을 C-minus의 reserved words들로 변경했다.

StateType state에 INEQ (=, == 구별), INLT(<, <= 구별), INNE(!=), INOVER(/, /* 구별), INCOMMENT_ (*/), 를 추가하고, := 토큰은 c-minus에서 지원하지 않으므로 INASSIGN은 삭제했다.

<getToken 함수>

INNUM과 INID는 기존 Tiny compiler와 동일하게 동작한다.

START state에서 =이 들어왔을 때 =(ASSIGN)인지, ==(EQ)인지 구별하기 위해 state는 INEQ이 되고, symbol 이므로 save = FALSE로 둔다. 그리고 이후 state가 INEQ일 때 받은 캐릭터가 = 이라면 ==(EQ) 이므로 currentToken = EQ이 되고, state = DONE이 되어 ==이 스캔된다. 그리고 만약 다른 문자라면 그냥 = 인 것이므로 ungetNextChar를 이용해서 input을 다시 되돌리고 currentToken = ASSIGN이 되어 =이 스캔된다.

<와 >일 때도 state가 각각 INLT, INGT로 바뀌며 INEQ와 동일하게 동작한다.

!이 들어오면 state는 INNE로 바뀌고, 이후 state가 INNE일 때 받은 input이 =이라면 currentToken = NE가 되고 != 이 스캔된다. 그런데 ! 뒤에는 항상 =이 와야하므로 만약 다른 문자라면 currentToken = ERROR가 되어 에러가 리턴된다.

/이 들어오면 /(OVER)인지 /*(COMMENT)인지 구분하기 위해 INOVER state로 변경된다.

그리고 INOVER state에서 들어온 input이 * 이라면 이후 input들은 주석이 되는 것이므로 INCOMMENT state로 변경된다. 만약 *이 아니라면 그냥 / (OVER) 인 것이므로 input을 다시 되돌려놓고 OVER을 스캔한다.

INCOMMENT state에서 *가 들어오면 주석이 끝나는 */ 을 의미하는 것인지 아니면 그냥 * 인지 구분하기 위해 INCOMMENT_ state로 들어간다.

INCOMMENT_ state에서 들어온 input이 / 이라면 주석이 끝난 것이므로 START state로 돌아간다. 그게 아니라면 아직 주석상태이므로 input을 되돌려놓고 INCOMMENT state로 돌아간다.

이외에 +, -, *, (,), ;, , , [,], {, } 이 들어오면 바로 그에 맞는 토큰을 currentToken으로 하고 state는 DONE이 되어 해당 토큰을 스캔한다.

START state에서 알파벳을 만나면 State가 INID로 들어가는데, 기존에는 이후에 알파벳이 아닌 것이 나오면 identifier 수용을 종료했는데, digit도 수용해야하므로 !isalpha© && !isdigit© 인 경우 수용을 종료한다.

Method2) tiny.l 파일을 cminus에 맞추어 수정한다.

Rule Section 부분에서 then, end, repeat, until, read, write의 tiny compiler에만 있는 토큰들을 삭제하고, else, while, return, int, void, =, ==, <=, >, >=, !=, [,], {, }, , 토큰을 rule로서 추가한다.

그리고 기존 tiny 에서 {} 였던 주석이 cminus에서는 /* */ 이므로, /* 에서는 */나 EOF를 만날 때 까지 input을 처리하도록 바꾼다.

기존 identifier는 letter로만 이루어진 것을 의미하므로 {letter}({digit}|{letter})* 로 변경하여 처음에만 letter이 나오고 이후에는 digit이 나와도 되도록 변경한다.

3. Example and Result Screenshot

Method1) 주어진 테스트 코드 test.1.txt 실행 결과

```
dayun@dayun-VirtualBox:~/2021_ele4029_2019056799/1_Scanner$ ./cminus_cimpl test.1.txt

C-MINUS COMPILATION: test.1.txt
        4: reserved word: int
        4: ID, name= gcd
        4: (
        4: reserved word: int
        4: ID, name= u
        4: ,
        4: reserved word: int
        4: ID, name= v
        4: )
        5: {
        6: reserved word: if
        6: (
        6: ID, name= v
        6: ==
        6: NUM, val= 0
        6: )
        6: reserved word: return
        6: ID, name= u
        6: ;
        7: reserved word: else
        7: reserved word: return
        7: ID, name= gcd
        7: (
        7: ID, name= v
        7: ,
        7: ID, name= u
        7: -
        7: ID, name= u
        7: /
        7: ID, name= v
        7: *
        7: ID, name= v
        7: )
        7: ;
        9: }
        11: reserved word: void
        11: ID, name= main
        11: (
        11: reserved word: void
        11: )
        12: {
        13: reserved word: int
```

```
        14: ID, name= x
        14: =
        14: ID, name= input
Software Updater
        14: )
        14: ;
        14: ID, name= y
        14: =
        14: ID, name= input
        14: (
        14: )
        14: ;
        15: ID, name= output
        15: (
        15: ID, name= gcd
        15: (
        15: ID, name= x
        15: ,
        15: ID, name= y
        15: )
        15: )
        15: ;
        16: }
        17: EOF
dayun@dayun-VirtualBox:~/2021_ele40
```

Method2) 주어진 테스트 코드 test.2.txt 실행 결과