Compiler Construction

Project #1: Scanner

제출일: 2021.10.31

컴퓨터소프트웨어학부

2019000982 이지원

Assignment

C-Minus Scanner를 C code와 Lex(flex) 두 가지의 방법으로 구현한다.

Environment

기본 환경: Mac OS

Lex(flex): Homebrew 3.2.17을 이용해 설치한 Flex 2.6.4 Apple(flex-34)

Implementation

1. C code - scan.c (DFA)

State: START, INASSIGN, INCOMMENT, INNUM, INID, DONE, INEQ, INLT, INGT, INNE, INOVER, INCOMMENT_

Reserved Words : globals.h에 추가해주고 scan.c에서 lookup table에 추가해준다.

+ - * ; , ( ) [ ] { } : 이 char들이 읽히면 바로 DONE state로 이동하고 현재 token이 각각 PLUS, MINUS, TIMES, SEMI, COMMA, LPAREN, RPAREN, LBRACE, RBRACE, LCURLY, RCURLY라고 표시한다.

< <= > >= == : 이 char들은 하나만 읽었을 때는 어떤 token인지 알 수 없으므로 INLT, INGT state로 이동하여 다음 char에 따라 DONE state로 이동하며 token을 LT, LE, GT, GE 중 해당하는 것으로 표시한다. <, >인 경우 읽었던 다음 char를 안 읽은 상태로 바꾼다.

!= : 이 char도 하나만 읽었을 때는 어떤 token인지 알 수 없다. 따라서 INNE state로 이동한 뒤 다음 token이 =인 경우에만 DONE state로 이동하고 token이 NE라고 표시한다. =이 아닌 경우 !라는 token은 없으므로 token을 ERROR라고 표시한 뒤 읽은 char를 안 읽은 상태로 바꾼다.

/* */ : 먼저 char에 /이 읽히면 INOVER state로 이동한다. INOVER state에서 뒤의 char를 읽었을 때 *이면 아니면 /이므로 DONE state로 이동하고 token을 OVER로 표시한다. 뒤의 char가 *이면 주석의 시작에 해당하므로 INCOMMENT state로 이동한다. INCOMMENT state에서는 뒤의 char가 *이 나오면 닫는 주석을 검사할 수 있는 INCOMMENT_ state로 이동한다. INCOMMENT_ state에서 뒤의 char로 /를 읽으면 주석이 끝난 것이므로 START state로 이동한다. /가 아니라면 *이 주석 안의 그냥 기호로 쓰인 것이므로 다시 INCOMMENT state로 이동한다.

ID : 원래 구현되어 있던 대로 구현하였다. alphabet이 읽히면 INID state로 이동하게 되고 INID state에서는 계속해서 뒤의 char를 읽는다. 만약 alphabet이 아닌 char를

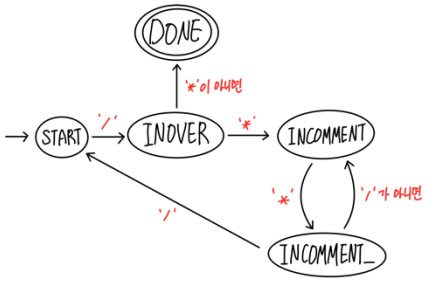
읽으면 해당 char를 안 읽은 상태로 바꾸고 DONE state로 이동하며 token을 ID로 표시한다.

NUM : 원래 구현되어 있던 대로 구현하였다. digit이 읽히면 INNUM state로 이동하게 되고 INNUM state에서는 계속해서 뒤의 char를 읽는다. 만약 digit이 아닌 char를 읽으면 해당 char를 안 읽은 상태로 바꾸고 DONE state로 이동하며 token을 NUM으로 표시한다.

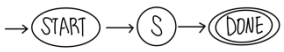
White Space : char가 ‘ ’, ‘\t’, ‘\n’인 경우에는 white space로 처리한다.

DFA : DFA는 모든 symbol에 대해 state로 이동해야 하지만 여기서는 간단히 필요한 symbol만을 이용해 간략한 DFA의 구조만을 그려보았다.

- 주석 처리의 DFA



- 나머지의 DFA



2. [Lex(flex)] - cminus.l (RE)

digit: [0-9]
number: (digit)+
letter [a-zA-Z]
identifier: (letter)(letter|digit)*
newline: \n
whitespace: [\t]+
위와 같이 Regular Expression을 정의한다.

Reserved Words: 각각 해당하는 Reserved Words를 return한다.

= < <= > >= == != + - * / ( ) [ ] { } ; , number identifier: 각각 해당하는 token을 return한다.

newine: lineno를 1 증가 시켜준다.

whitespace: 아무것도 하지 않는다.

/* */: /*를 읽으면 그 뒤의 char를 하나씩 읽으면 이전의 char를 c1에 저장한다. 즉, c1, c2 순서대로 바로 뒤 두개의 char를 연속적으로 읽어서 저장한다. 읽는 중 \n이 나오면 줄이 바뀐 것이므로 lineno를 1씩 증가시켜 준다. 만약 c1, c2가 각각 *, /로 읽힌다면 주석이 끝나는 것을 의미하므로 while문을 멈춘다. 주석으로 따로 출력할 것이 없으므로 특별히 다른 token을 return 해주지 않는다.

3. **util.c**

printToken() 함수에서 필요한 부분을 수정한다.

Reserved Words를 token으로 받은 경우 reserved word: \n의 형식으로 각각의 reserved word를 출력한다.

ASSIGN, EQ, NE, LT, LE, GT, GE, LPAREN, RPAREN, LBRACE, RBRACE, LCURLY, RCURLY, SEMI, COMMA, PLUS, MINUS, TIMES, OVER, ENDFILE, NUM, ID, ERROR인 경우 각각 해당하는 token을 출력해준다.

Operation

1. **C code – DFA**

   cminus_cimpl을 이용해 test file들을 실행 시키면 c code를 이용해 만든 lexer가 test file을 tokenize 한다. c code로 만든 lexer는 scan.c에 정의한 DFA에 의해 작동한다. START state에서 시작해 DONE state에 가기 전까지 while문을 계속 돌아가며 char를 하나씩 읽어서 알맞은 state로 이동한다. DONE state에 가면 getToken 함수가 끝나고 현재 token을 return 한다. main.c에서 getToken을 진행하여 token을 알아내고 이를 util.c에 있는 print 함수를 이용해 출력한다.

2. **LEX (flex) – RE**

   cminus_lex를 이용해 test file들을 실행 시키면 cminus.l에 Regular Expression으로 정의해 둔 spec에 맞게 자동으로 getToken 함수가 생성되고 lexer가 생성된다. 그리고 이를 이용해 test file들을 tokenize 해서 print 하게 된다.

Execution Method

1. 1_Scanner 폴더 안의 파일들을 같은 위치에 저장한다.
2. 터미널에서 make 명령어를 통해 make clean 후 make를 진행한다.
3. cminus_cimpl과 cminus_lex를 원하는 test file과 함께 실행시킨다.
   - 출력 결과를 새로운 파일에 작성하여 확인하고 싶으면 > [file name]을 붙인다.
   - Ex. ./cminus_cimple test.1.txt
   - Ex. ./cminus_lex test.1.txt > result1.txt

Result

test.1.txt와 이를 c code, lex 두 가지 방법으로 실행한 결과의 사진을 각각 첨부하였다.

- test.1.txt

test.1.txt

```
/* A program to perform Euclid's
   Algorithm to computer gcd */

int gcd (int u, int v)
{
        if (v == 0) return u;
        else return gcd(v,u-u/v*v);
        /* u-u/v*v == u mod v */
}

void main(void)
{
        int x; int y;
        x = input(); y = input();
        output(gcd(x,y));
}
```

- ./cminus_cimpl test.1.txt

```
leejiwon@MacBook-Pro 1_Scanner % ./cminus_lex test.1.txt

TINY 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
       13: ID, name= x
       13: ;
       13: reserved word: int
       13: ID, name= y
       13: ;
       14: ID, name= x
       14: =
       14: ID, name= input
       14: (
       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
```

- ./cminus_lex test.1.txt

```
leejiwon@MacBook-Pro 1_Scanner % ./cminus_cimpl test.1.txt

TINY 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
       13: ID, name= x
       13: ;
       13: reserved word: int
       13: ID, name= y
       13: ;
       14: ID, name= x
       14: =
       14: ID, name= input
       14: (
       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
```