Project #1. Scanner

컴퓨터소프트웨어학부
2018008722 유수영

1. Compile 환경 및 방법

➔ Make로 컴파일

```
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/1_Scanner$ ls
analyze.c  cminus.l   lex       parse.h     scan.c    test.1.txt  util.c
analyze.h  code.c     main.c    readme.unx  scan.h    test.2.txt  util.h
cgen.c     code.h     Makefile  sample.tm   symtab.c  test.txt    yacc
cgen.h     globals.h  parse.c   sample.tny  symtab.h  tm.c
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/1_Scanner$ make
gcc  -c main.c
main.c:48:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main( int argc, char * argv[] )
 ^~~~
gcc  -c util.c
util.c:124:8: warning: type defaults to ‘int’ in declaration of ‘indentno’ [-Wimplicit-int]
 static indentno = 0;
        ^~~~~~~~
gcc  -c scan.c
gcc  -c parse.c
parse.c:23:19: warning: conflicting types for built-in function ‘exp’ [-Wbuiltin-declaration-mis
match]
 static TreeNode * exp(void);
                   ^~~
gcc  -c symtab.c
gcc  -c analyze.c
gcc  -c code.c
gcc  -c cgen.c
gcc  main.o util.o scan.o parse.o symtab.o analyze.o code.o cgen.o -o scanner_cimpl
flex cminus.l
gcc  -c lex.yy.c
gcc  main.o util.o lex.yy.o parse.o symtab.o analyze.o code.o cgen.o -o scanner_flex -lfl
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/1_Scanner$ ls
analyze.c  cgen.o    globals.h  main.o    readme.unx  scanner_cimpl  symtab.o    util.c
analyze.h  cminus.l  lex        Makefile  sample.tm   scanner_flex   test.1.txt  util.h
analyze.o  code.c    lex.yy.c   parse.c   sample.tny  scan.o         test.2.txt  util.o
cgen.c     code.h    lex.yy.o   parse.h   scan.c      symtab.c       test.txt    yacc
cgen.h     code.o    main.c     parse.o   scan.h      symtab.h       tm.c
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/1_Scanner$
```

➔ Make 명령어를 통해 Makefile(블랙보드에 올라와 있는 Makefile 그대로 사용)의 내용들이 실행되어 실행파일 scanner_cimpl과 scanner_flex가 생성된 것을 볼 수 있다.

2. C-scanner 코드 작성 및 tiny compiler 코드 변경 내용

<Globals.h>

1) MAXRESERVED의 값을 tiny의 8에서 6으로 변경(if, else, while, return, int, void의 6가지 reserved words만을 c-minus에서 사용하기 때문)

2) TokenType을 c-minus에 맞추어 변경한다. Reserved word는 if, else, while, return, int, void로, special symbols은 c-minus가 처리하는 ASSIGN, EQ, NE, LT, LE, GT, GE, PLUS, MINUS, TIMES, OVER, LPAREN, RPAREN, LBRACE, RBRACE, LCURLY, RCURLY, SEMI, COMMA로 변경해준다.

<Main.c>

1) scanner만을 만드는 과제이기 때문에 NO_PARSE를 TRUE로, 각 txt의 줄들을 출력하기 위해 EchoSource를 TRUE로 바꾸어 준다. TraceScan도 TRUE로 바꾸어 준다.

<scan.c>

1) dfa의 state를 정의해준다.

1- START : 들어오는 token이 처음으로 시작하는 state. Initial state.
2- INEQ : '='를 나타내는 state로, START에서 '='가 들어왔을 때 INEQ라는 state로 transition이 이루어진다.
3- INCOMMENT : comment를 나타내기 위한 state로, '/ ' 뒤에 '*'가 들어왔을 때 INOVER에서 INCOMMENT로 transition이 이루어진다.
4- INNUM : 숫자를 나타내는 state로, START에서 어느 digit가 들어왔을 때 INNUM으로 transition이 이루어진다.
5- INID : identifier(변수)를 나타내는 state로, START에서 alphabet이 들어오면 INID로 transition이 진행된다.
6- DONE : 마지막 state로, 모든 state에서 끝날 때는 무조건 DONE으로 들어온다.
7- INLT : less than(<)을 나타내기 위한 state로, START에서 <가 들어오면 INLT로 transition이 이루어진다.
8- INGT : greater than(>)을 나타내기 위한 state로, START에서 >가 들어오면 INGT로 transition이 이루어진다.
9- INNE : '!='을 나타내기 위해서 사용하는 state로, START에서 '!'가 들어오면 INNE로 transition하고, 바로 뒤에 '='가 오는지를 확인하게 된다. ('='가 들어오면 accept, 들어오지 않으면 error가 된다. 단, error가 발생해도 원활한 프로그램 진행을 위해 DONE state로 보내되, currentToken에 error를 부여한다.)
10- INOVER : '/ '를 나타내는 state이다. START에서 '/ '가 들어오면 INOVER로 state transition이 이루어진다.

2) Scan.c의 reserved word도 c-minus의 reserved word로 변경해준다.

3) 각 문자들을 scan하는 getToken함수의 while문은 state가 DONE에 도착하여 token이 accept되었거나, INCOMMENT에 도착하여 accept될 수 없는 경우 빠져나가게 된다. ERROR를 제외한 나머지 token들은 다양한 state에 있다가 무조건 DONE으로 돌아오게 되고, comment는 token으로 인식되지 않아야 하기 때문에 DONE state가 아니라 INCOMMENT라는 단독 state로 transition하게 되고, 이는 accept되지 않아 유효한 token이라고 인식하지 못하도록 하는 개념으로 생각할 수 있다.

4) START state에서의 transition

1- 한 번의 transition을 거치고도 DONE으로 가지 않고 새로운 transition으로 이동하는 경우 : 이는, 두 번 이상의 transition을 통해 DONE으로 가는 경우로 multicharacter로 이루어진 token들, 두개의 문자로 이루어진 symbol, 그리고 comment를 처리하는 부분이다.

들어오는 character가 digit인 경우, 그 뒤로 계속 숫자가 들어오면 여러 자리 수의 number가 되므로 INNUM이라는 state로 transition해주고, 다음 들어오는 character를 다시 볼 수 있도록 한다.

들어오는 character가 alphabet인 경우, 그 뒤로 어떤 알파벳이 더 오느냐에 따라 reserved word가 될 수도, identifier가 될 수도 있으므로 INID state로 transition하고, 계속해서 다음 character를 확인할 수 있도록 한다.

들어오는 character가 '=' | '/' | '>' | '<' 인 경우, 다음에 '=' | '*' | '=' | '=' 가 오면 current token의 상태가 EQ | comment | INGE | INLE가 되고, 그 외의 다른 상관없는 문자가 오면 current token의 상태가 ASSIGN | OVER | INGT | INLT가 될 수도 있기 때문에 먼저 INEQ | INOVER | INGT | INLT라는 state로 transition하고, 다음에 들어오는 문자를 확인하도록 한다.

2- 한 번의 transition으로 DONE으로 transition 하는 경우 : 단일 문자로 이루어진 symbol들을 처리하는 부분으로 state를 DONE으로 transition하고, 각 문자가 + / - / * / { / } / [/] / (/) / ; / , 인 경우 current token을 PLUS / MINUS / TIMES / LCURY / RCURLY / LBRACE / RBRACE / LPAEN / RPAREN / SEMI / COMMA로 설정해준다.

5) START state외의 다른 state에서 transition 하는 경우

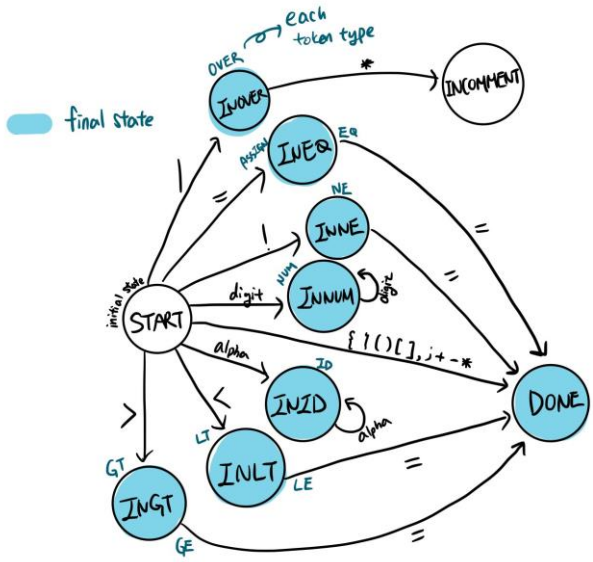
1- INNUM과 INID인 경우 : 다음으로 들어온 char가 digit / alphabet이면 계속해서 state를 유지하고, 아니라면 DONE으로 보낸 뒤 그 앞까지의 token을 NUM / ID로 설정한다.

2- INEQ / INLT / INGT / INNE 인 경우 : 각 state에 있을 때, 모두 DONE으로 transition하고, 들어오는 char가 = 인 경우 각각 currentToken은 EQ, LE, GE, NE로, =가 아닌 경우에는 ASSIGN, LT, GT, ERROR로 설정한다.

3- INOVER인 경우 : /다음으로 들어오는 것이 '*'라면 comment를 나타내는 것이기 때문에 그 뒤 내용들은 '*/'가 나오기 전까지 모두 token으로 처리하지 않도록 '*/'가 나올 때까지 다음 char들을 계속해서 읽는다. 이때 isComment의 값을 1로 바꿔준다. 그러나 '*'가 아니라면 DONE으로 보내고, currentToken을 OVER로 설정한다.

6) isComment 변수를 하나 추가하여, 해당 string이 comment라고 판단되면, isComment의 값이 1로 바뀌고, comment는 token의 종류가 정의되지 않기 때문에 printToken이 실행되지 않도록 한다.

3. 구현한 DFA



➔ code상에서는 while문의 편의성을 위해 comment를 제외한 모든 token이 DONE으로 가는데, DFA에서는 각각이 final state가 되어 개념적으로 DFA를 표현하면 위와 같게 나온다.

4. Flex를 이용하기 위한 cminus.l

➔ Tiny.l 파일을 조금 수정하여 작성. If, else, while, return, int, void의 reserved word와 =, ==, !=, <, <=, >, >=, +, -, *, /, (,), [,], {, }, ;, ,, /* 순으로 symbol들을, 그리고 number, identifier, newline, whitespace들을 rules section에 정의해준다. 모두 각 TokenType들을 return해주지만, new line은 line no.를 하나 증가해주고, whitespace는 무시한다. /*(comment)의 경우 주석이 끝났음을 알리는 '*/'이 나올 때까지 계속해서 char를 읽고,(그냥 읽기만 함으로써 token으로 분류하지 않음) 만약, new line을 읽으면 line의 no.도 하나씩 증가시켜준다.

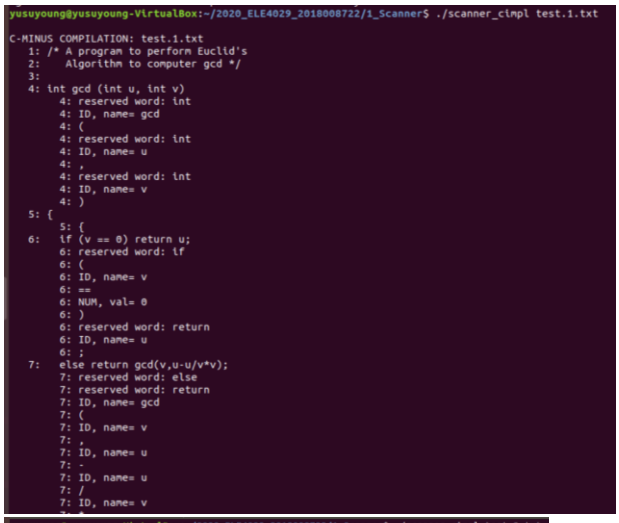
5. 실행 화면

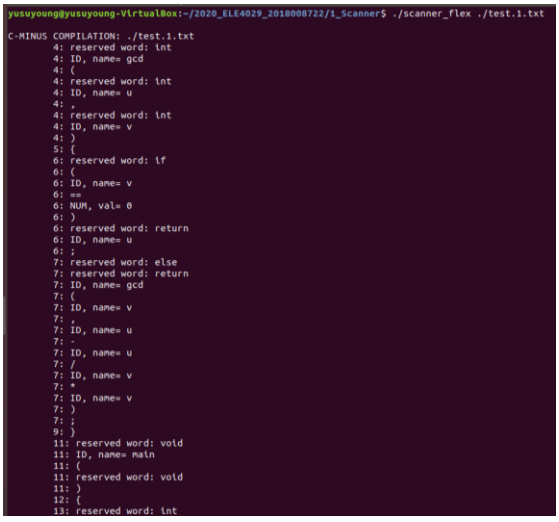
```
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/1_Scanner$ cat test.txt
void main(){
        /* This is test comment that is nested comment /* this is nested comment */ test*/
}
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/1_Scanner$ ./scanner_cimpl test.txt

C-MINUS COMPILATION: test.txt
   1: void main(){
        1: reserved word: void
        1: ID, name= main
        1: (
        1: )
        1: {
   2:   /* This is test comment that is nested comment /* this is nested comment */ test*/
        2: ID, name= test
        2: *
        2: /
   3: }
        3: }
        4: EOF
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/1_Scanner$ ./scanner_flex ./test.txt

C-MINUS COMPILATION: ./test.txt
        1: reserved word: void
        1: ID, name= main
        1: (
        1: )
        1: {
        2: ID, name= test
        2: *
        2: /
        3: }
        4: EOF
```

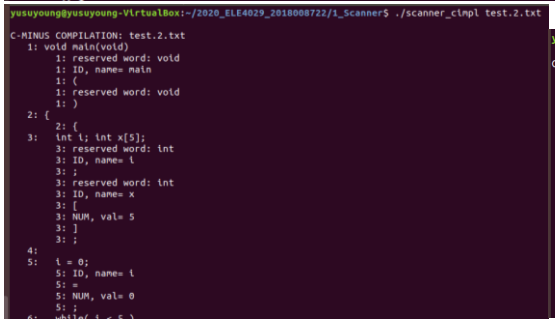
➔ Nested comment인 경우, comment 안의 comment 기호(/*)는 무시하고 */가 나오면 다음은 comment가 아닌 일반 명령어로 인식한다.

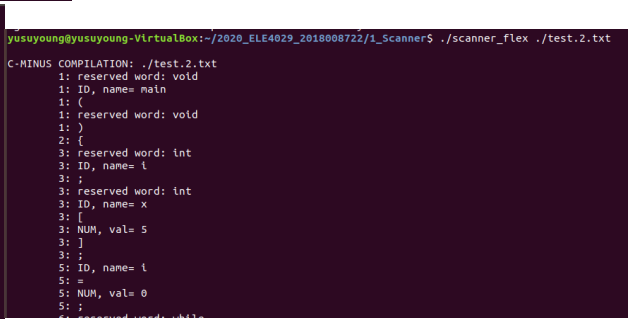
```
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/1_Scanner$ cat test.txt
void main(){
        / *
}
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/1_Scanner$ ./scanner_cimpl test.txt

C-MINUS COMPILATION: test.txt
   1: void main(){
        1: reserved word: void
        1: ID, name= main
        1: (
        1: )
        1: {
   2:   / *
        2: /
        2: *
   3: }
        3: }
        4: EOF
yusuyoung@yusuyoung-VirtualBox:~/2020_ELE4029_2018008722/1_Scanner$ ./scanner_flex ./test.txt

C-MINUS COMPILATION: ./test.txt
        1: reserved word: void
        1: ID, name= main
        1: (
        1: )
        1: {
        2: /
        2: *
        3: }
        4: EOF
```

➔ comment기호인 /* 사이에 공백이 있다면(/ *의 경우) comment로 인식하지 않고 각각을 over와 times로 인식한다.









➔ 예시였던 test.1.txt과 test.2.txt가 scanner와 flex에서 잘 돌아가는 것을 확인할 수 있다.