

Compiler Project 1_Scanner

2019062833 김유진

1. 컴파일 방법

```
Makefile
1  # Makefile for C-Minus Scanner
2  # ./lex/tiny.l --> ./cminus.l
3
4  CC = gcc
5
6  CFLAGS = -W -Wall
7
8  OBJS = main.o util.o scan.o
9  OBJS_LEX = main.o util.o lex.yy.o
10
11 .PHONY: all clean
12 all: cminus_cimpl cminus_lex
13
14 clean:
15     -rm -vf cminus_cimpl cminus_lex *.o lex.yy.c
16
17 cminus_cimpl: $(OBJS)
18     $(CC) $(CFLAGS) -o $@ $(OBJS)
19
20 cminus_lex: $(OBJS_LEX)
21     $(CC) $(CFLAGS) -o $@ $(OBJS_LEX) -lf1
22
23 main.o: main.c globals.h util.h scan.h
24     $(CC) $(CFLAGS) -c -o $@ $<
25
26 scan.o: scan.c globals.h util.h scan.h
27     $(CC) $(CFLAGS) -c -o $@ $<
28
29 util.o: util.c globals.h util.h
30     $(CC) $(CFLAGS) -c -o $@ $<
31
32 lex.yy.o: lex.yy.c globals.h util.h scan.h
33     $(CC) $(CFLAGS) -c -o $@ $<
34
35 lex.yy.c: cminus.l
36     flex -o $@ $<
37
38
```

1) **C code**: make 명령어 사용으로 cminus_cimpl 프로그램 생성 후 실행

2) **Lex**: flex cminus.l 명령 후 생성 되는 lex.yy.c 파일을 사용. make 명령어로 cminus_lex 프로그램 생성 후 실행

2. 개발 환경

- Ubuntu 20.04

3. 구현과 작동

이번 프로젝트에서는 공통적으로 main.c, globals.h, utils.c 의 일부 코드를 수정한 후 C code 구현 방식에서는 scan.c를 수정하고, Lex 방식에서는 cminus.l 파일을 생성하여 C-Minus Scanner를 구현하였다.

1) C code – scan.c

DFA로 구현이 되는 방식으로 아래와 같이 state 정의를 하게 된다.

```
/* states in scanner DFA */
typedef enum
{
    START, INASSIGN, INCOMMENT, INCOMMENT_, INNUM, INID, DONE,
    INEQ, INNE, INLT, INGT, INOVER }
    StateType;
```

getToken 함수에서 DFA를 구현하여 token을 인식하게 된다. 우선 처음 state는 start이며 첫 입력 글자에 따라 INNUM, INID, INASSIGN, INOVER, INNE, INGT 상태로 이동하게 되며 여기에 모두 해당되지 않는 경우 바로 Done state로 이동하게 되며 토큰이 결정된다.

Done을 제외한 다른 state로 이동한 것은 한 글자만 읽어서는 토큰이 결정될 수 없는 경우임을 의미한다. < 와 <=, ==과 = 등의 경우처럼 두번째 글자를 확인하게 되면 토큰이 결정된다. 만약 <=가 아니라 < 였다면 방금 읽은 글자는 ungetNextChar() 함수를 사용하여 되돌린다.

Done 상태가 되기전까지 tokenString에 input 글자를 저장하고 있다가 Done state로 가게 되면 현재 토큰을 결정하고 출력하게 된다.

2) Lex – cminus.l

우선 기존의 tiny.l 코드를 복사해온 뒤 수정하여 구현한다. 수정해야할 부분은 크게 3가지로 나뉘는데 정의 구역, 규칙 구역, Subroutine 구역이다.

정의 구역에는 Rex naming을 해주게 되는데 아래사진과 같이 regular 표현식을 통해서 정의를 하게 된다.

```
digit      [0-9]
number     {digit}+
letter     [a-zA-Z]
identifier {letter}{letter|{digit}}*
newline    \n
whitespace [ \t]+
```

이렇게 정의된 표현들을 사용해서 규칙 구역을 구현한다. 아래와 같이 키워드와 Symbol들에 대한 규칙을 정의해준다. 이때 반환하게 되는 것은 globals.h에 정의해둔 TokenType이다. input()을 사용해서 한 글자씩 받아오며 unput(c)(scan.c에서의 ungetNextChar())을 통해서 되돌린다.

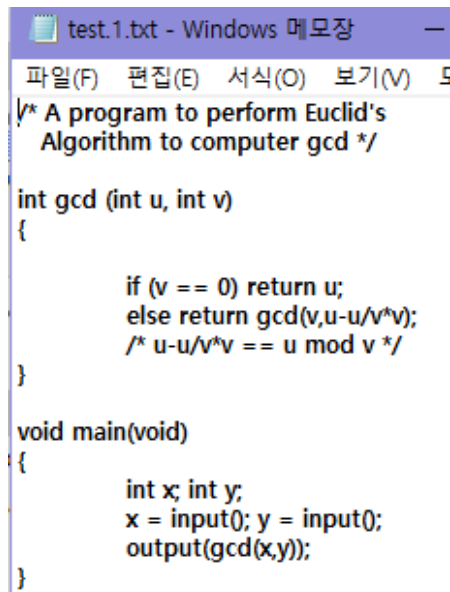
```
"if"      {return IF;}
"else"    {return ELSE;}
"while"   {return WHILE;}
"return"  {return RETURN;}
"int"     {return INT;}
"void"    {return VOID;}
```

```
"["       {return LBRACE;}
"]"       {return RBRACE;}
"{"       {return LCURLY;}
"}"       {return RCURLY;}
"("       {return LPAREN;}
")"       {return RPAREN;}
```

```
"<"      {
    char c=input();
    if(c=='=') return LE;
    else {
        unput(c);
        return LT;
    }
};
```

Subroutine 구역은 유저가 정의하는 추가적으로 사용할 함수가 존재하는 곳으로 기본적으로 getToken() 함수가 존재한다. 타겟 소스를 받아서 정의 구역과 규칙 구역의 정보를 토대로 토큰을 생성하고 출력하게 된다.

4. 예제와 결과



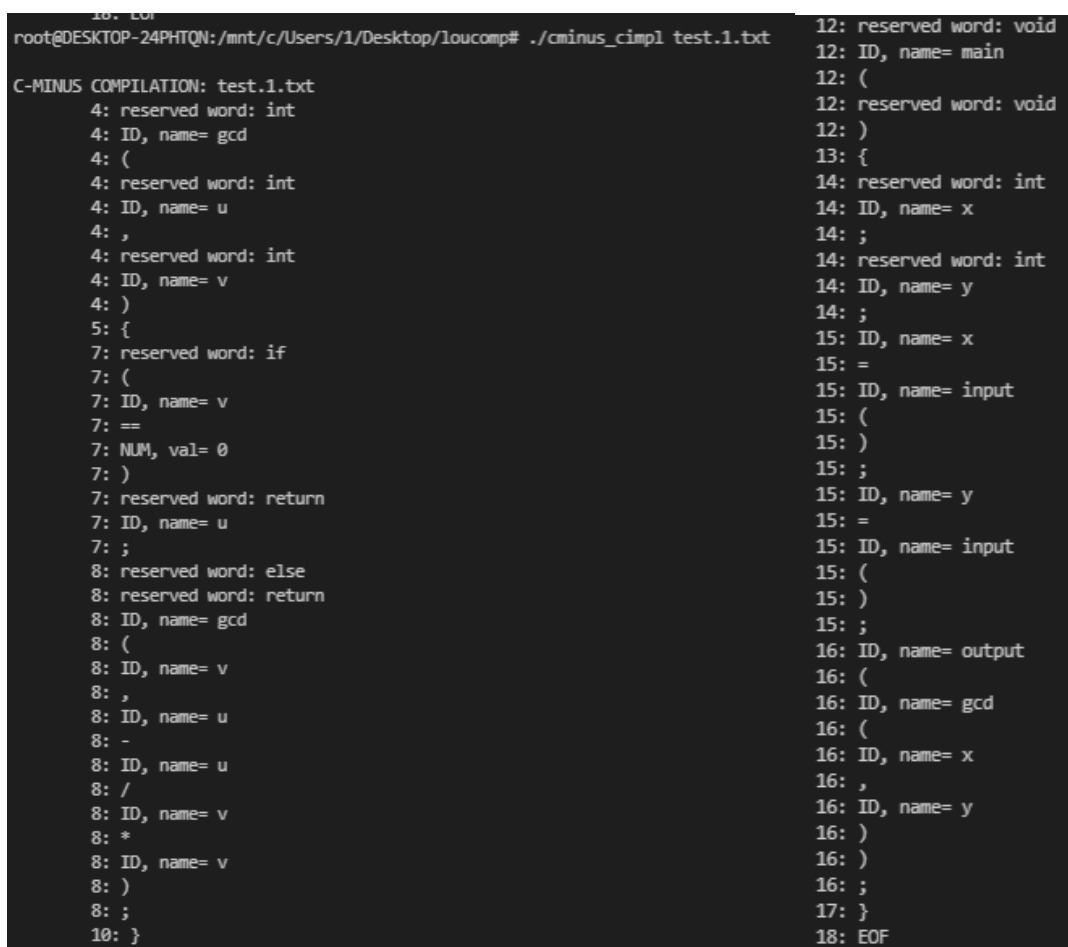
```
test.1.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도구(D)
/* A program to perform Euclid's
   Algorithm to computer gcd */

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));
}
```

1) test.1.txt

2) cminus_cimpl 결과



```
root@DESKTOP-24PHTQN:/mnt/c/Users/1/Desktop/loucomp# ./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: {
7: reserved word: if
7: (
7: ID, name= v
7: ==
7: NUM, val= 0
7: )
7: reserved word: return
7: ID, name= u
7: ;
8: reserved word: else
8: reserved word: return
8: ID, name= gcd
8: (
8: ID, name= v
8: ,
8: ID, name= u
8: -
8: ID, name= u
8: /
8: ID, name= v
8: *
8: ID, name= v
8: )
8: ;
10: }
12: reserved word: void
12: ID, name= main
12: (
12: reserved word: void
12: )
13: {
14: reserved word: int
14: ID, name= x
14: ;
14: reserved word: int
14: ID, name= y
14: ;
15: ID, name= x
15: =
15: ID, name= input
15: (
15: )
15: ;
15: ID, name= y
15: =
15: ID, name= input
15: (
15: )
15: ;
16: ID, name= output
16: (
16: ID, name= gcd
16: (
16: ID, name= x
16: ,
16: ID, name= y
16: )
16: )
16: ;
17: }
18: EOF
```

3) cminus_lex 결과

```
root@DESKTOP-24PHTQN:/mnt/c/Users/1/Desktop/loucomp# ./cminus_lex 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: {
7: reserved word: if
7: (
7: ID, name= v
7: ==
7: NUM, val= 0
7: )
7: reserved word: return
7: ID, name= u
7: ;
8: reserved word: else
8: reserved word: return
8: ID, name= gcd
8: (
8: ID, name= v
8: ,
8: ID, name= u
8: -
8: ID, name= u
8: /
8: ID, name= v
8: *
8: ID, name= v
8: )
8: ;
10: }
12: reserved word: void
12: ID, name= main
12: (
12: reserved word: void
12: )
13: {
14: reserved word: int
14: ID, name= x
14: ;
14: reserved word: int
14: ID, name= y
14: ;
15: ID, name= x
15: =
15: ID, name= input
15: (
15: )
15: ;
15: ID, name= y
15: =
15: ID, name= input
15: (
15: )
15: ;
16: ID, name= output
16: (
16: ID, name= gcd
16: (
16: ID, name= x
16: ,
16: ID, name= y
16: )
16: )
16: ;
17: }
18: EOF
```