

2019032160 유석원

위의 규칙에 맞춰 먼저 util.c 파일을 수정했습니다. 해당 파일은 토큰 출력에 대한 함수인 printToken 함수가 있으므로, 규칙에 맞춰 수정을 해주었습니다. 다음으로 scan.c 파일에서도 마찬가지로 예약어에 관련된 변수를 수정했습니다. Scanner의 메인 로직은 getToken 함수에서 이루어지므로, 해당 함수의 로직을 수정했습니다. C 코드의 경우, 한 글자씩 입력을 읽어와 state를 확인하면서 token을 식별하기 때문에 DFA가 필수적입니다. 다만, DFA로 그릴 경우, 너무나도 많은 state를 그려야 하므로 간단한 NFA를 그려 표현하고, 코드로 바꾸었습니다.



```

cactus@cactus:~/compiler/CompilerDesign/Project1/files$ ./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

```

```

cactus@cactus:~/compiler/CompilerDesign/Project1/files$ ./cminus_cimpl test.2.txt

TINY COMPILATION: test.2.txt
1: reserved word: void
1: ID, name= main
1: (
1: reserved word: void
1: )
2: {
3: reserved word: int
3: ID, name= i
3: ;
3: reserved word: int
3: ID, name= x
3: [
3: NUM, val= 5
3: ]
3: ;
5: ID, name= i
5: =
5: NUM, val= 0
5: ;
6: reserved word: while
6: (
6: ID, name= i
6: <
6: NUM, val= 5
6: )
7: {
8: ID, name= x
8: [
8: ID, name= i
8: ]
8: =
8: ID, name= input
8: (
8: )
8: ;
10: ID, name= i
10: =
10: ID, name= i
10: +
10: NUM, val= 1
10: ;
11: }
13: ID, name= i
13: =
13: NUM, val= 0
13: ;
14: reserved word: while
14: (
14: ID, name= i
14: <=
14: NUM, val= 4
14: )
15: {
16: reserved word: if
16: (
16: ID, name= x
16: [
16: ID, name= i
16: ]
16: !=
16: NUM, val= 0
16: )
17: {
18: ID, name= output
18: (
18: ID, name= x
18: [
18: ID, name= i
18: ]
18: )
18: ;
19: }
20: }
21: }
22: EOF

```

## Lex 코드

Lex 코드는 `globals.h`, `util.h`, `main.c`를 사용하고, `scan.c` 파일은 사용하지 않습니다. Lex 내부의 `%% //code %%` 사이에 `lex rule`이 정의되어 해당 `rule`이 `scan.c`의 `getToken` 함수와 같은 역할을 하게 됩니다. 주석을 제외하고는 두 글자도 입력을 받아들 수 있기 때문에 예약어와 모든 `symbol`을 저장해주고, `newline`은 `lineno`를 하나 증가시키고, `whitespace`는 무시하며, 주석의 시작인 `/*`가 입력되면 주석을 닫는 `*/`를 받을 때까지 한 글자씩 입력을 읽어오는 반복문을 실행합니다. 주석이 닫히기 전까지는 계속 무시하며,

그러다 주석이 닫히지 않은 상태로 EOF에 도달할 수 있기 때문에 읽은 문자가 EOF나 'W0'가 되면 반복문을 빠져나옵니다.

```
cactus@cactus:~/compiler/CompilerDesign/Project1/files$ ./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

cactus@cactus:~/compiler/CompilerDesign/Project1/files$ ./cminus_lex test.2.txt
TINY COMPILATION: test.2.txt
1: reserved word: void
1: ID, name= main
1: (
1: reserved word: void
1: )
2: {
3: reserved word: int
3: ID, name= i
3: ;
3: reserved word: int
3: ID, name= x
3: [
3: NUM, val= 5
3: ]
3: ;
5: ID, name= i
5: =
5: NUM, val= 0
5: ;
6: reserved word: while
6: (
6: ID, name= i
6: <
6: NUM, val= 5
6: )
7: {
8: ID, name= x
8: [
8: ID, name= i
8: ]
8: =
8: ID, name= input
8: (
8: )
8: ;
10: ID, name= i
10: =
10: ID, name= i
10: +
10: NUM, val= 1
10: ;
11: }
13: ID, name= i
13: =
13: NUM, val= 0
13: ;
14: reserved word: while
14: (
14: ID, name= i
14: <=
14: NUM, val= 4
14: )
15: {
16: reserved word: if
16: (
16: ID, name= x
16: [
16: ID, name= i
16: ]
16: !=
16: NUM, val= 0
16: )
17: {
18: ID, name= output
18: (
18: ID, name= x
18: [
18: ID, name= i
18: ]
18: )
18: ;
19: }
20: }
21: }
22: EOF
```

## 빌드

PPT의 설명을 따라 main.c 파일에서 NO\_PARSE와 TraceScan을 TRUE로 설정했고, EchoSource는 FALSE로 설정했습니다. Makefile이 제공되기 때문에 해당하는 makefile과 test/result 텍스트 파일을 기존의 컴파일러 파일들이 있는 디렉토리에 함께 뒀으며, 모든

결과를 확인한 뒤 현재 “make clean” 명령어를 실행해 오브젝트 파일과 cminus\_cimpl, cminus\_lex 파일은 삭제된 상태입니다.

LMS에 업로드되어 있는 zip 파일은 Mac에서 압축한 파일이며, 혹시 오류로 인해 압축이 해제되지 않거나 압축을 해제하고 파일이 존재하지 않는 경우 연락해 주시기 바랍니다.