

# Project #1. Scanner

컴퓨터소프트웨어학부 2017029589 류지범

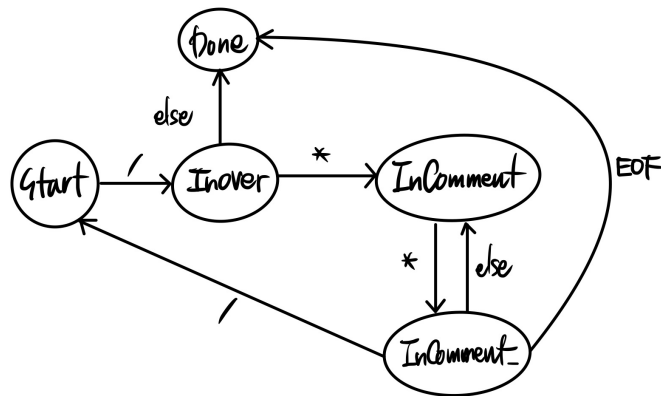
## #0 공통

- Environment
  - Ubuntu 18.04.02 LTS
- main.c
  - `NO_PARSE` 를 `TRUE` 로 고치고 `TraceScan` 를 `TRUE` 로 고쳤다.
- globals.h
  - `MAXRESERVED` 를 8에서 6으로 수정했다.
  - reserved words로 사용되지 않는 `THEN`, `END`, `REPEAT`, `UNTIL`, `READ`, `WRITE` 를 지우고 `WHILE`, `RETURN`, `INT`, `VOID` 를 추가했다.
  - special symbols로 추가적으로 사용되는 `LE`, `GT`, `GE`, `NE`, `LBRACE`, `RBRACE`, `LCURLY`, `RCURLY`, `COMMA` 를 추가했다.
- util.h
  - reserved word인 `IF`, `ELSE`, `WHILE`, `RETURN`, `INT`, `VOID` 에 대해 `fprintf(listing, "reserved word: %s\n", tokenString);` 을 출력하도록 해줬다.
  - 추가된 special symbols인 `LE`, `GT`, `GE`, `NE`, `LBRACE`, `RBRACE`, `LCURLY`, `RCURLY`, `COMMA` 에 대해 각각 `"<=\n"`, `">\n"`, `">=\n"`, `"!=\n"`, `"[\n"`, `"]\n"`, `"{\n"`, `"}\n"`, `",\n"` 을 출력하도록 추가해줬다.
  - `ASSIGN` 은 `":=\n"` 에서 `"=\n"` 으로 수정했다.
- 실행
  - Makefile로 컴파일한 후 다음과 같이 실행한다.
  - method1
    - `./cminus_cimpl testfile.txt`
  - method2
    - `./cminus_lex testfile.txt`

## #1 cminus\_cimpl

- `cminus_cimpl` 은 `cminus` 의 문법에 맞는 DFA를 찾고, `scan.c` 의 `getToken(void)` 를 수정해서 구현한다.
- 숫자가 들어왔을 경우엔 `TINY` 와 똑같이 처리한다.
- 알파벳이 들어왔을 경우엔 `INID` state로 간다.
  - 변수는 알파벳으로 시작하고 그 이후엔 알파벳과 숫자 어느 조합도 가능하므로 `INID` 에서 `!(isalpha(c) || isdigit(c))` 일 경우에만 `DONE` 으로 보내도록 한다.
- `cminus`에선 `'='` 이 `ASSIGN` 이기 때문에 `'='` 이 들어왔을 때 `INASSIGN` state를 가지도록 한다.
  - 그 후에 `'='` 이 이어서 들어오면 Equal을 뜻하므로 `currentToken` 에 `EQ` 를 대입한다. 그렇지 않을 경우엔 `ASSIGN` 을 대입하고 `ungetNextChar()` 를 한다.
- `'>'` 이 들어왔을 땐 `">="` 이 나올 수 있으므로 `INGT(greater than)` state를 가지도록 한다.
  - `INGT` 에서 `'='` 이 들어왔을 땐 `currentToken` 에 `GE` 를 대입해준다. 그렇지 않을 경우 `GT` 를 대입한다.

- '<' 이 들어왔을 땐 "<=" 이 나올 수 있으므로 INLT(less than) state를 가지도록 한다.
  - INLT 에서 '=' 이 들어왔을 땐 currentToken 에 LE 를 대입해준다. 그렇지 않을 경우 LT 를 대입한다.
- '!' 이 들어왔을 땐 "!=" 이 나올 수 있으므로 INNE(not equal) state를 가지도록 한다.
  - INNE 에서 '=' 이 들어왔을 땐 currentToken 에 NE 를 대입해준다. 그렇지 않을 경우 없는 토큰이므로 ERROR 를 대입한다.
- cminus에선 "/"\* 을 주석으로 가지기 때문에 '/' 이 들어오면 INOVER state를 가지도록 한다.
- '(', ')', '[', ']', '{', '}', ',', ' ' 이 들어오면 각각 해당하는 special symbol 을 currentToken 으로 가지도록 한다.
- DFA of Comment
  -



- '/' 이 나오면 INOVER 로 가고 그 후에 '\*' 이 나오지 않을 경우엔 OVER 로 취급하고 DONE 으로 간다. '\*' 이 나올 경우엔 주석 상태로 진입하고 INCOMMENT 로 간다.
- 그 후에 '\*' 이 나오면 주석이 끝날 가능성이 있으므로 INCOMMENT\_ 로 보낸다. EOF 가 나올 수 있으므로 체크해준다.
- '\*' 후에 바로 '/' 이 나오면 주석이 닫히는데 그렇지 않을 경우 주석안에 포함된 내용으로 간주하면 되므로 다시 INCOMMENT 로 보내서 '\*' 이 다시 나올 때까지 돌도록 한다.
- 코드는 다음과 같다.

```

case INOVER:
    save = FALSE;
    if (c == '*') {
        state = INCOMMENT;
    }
    else {
        ungetNextChar();
        currentToken = OVER;
        state = DONE;
    }
    break;
case INCOMMENT:
    save = FALSE;
    if (c == EOF)
    {
        state = DONE;
    }

```

```

        currentToken = ENDFILE;
    }
    if (c == '*') {
        state = INCOMMENT_;
    }
    break;
case INCOMMENT_:
    save = FALSE;
    if (c == EOF)
    {
        state = DONE;
        currentToken = ENDFILE;
    }
    else if (c == '/') state = START;
    else if (c == '*') state = INCOMMENT_;
    else state = INCOMMENT;
    break;
case INASSIGN:
    state = DONE;
    if (c == '=')
        currentToken = EQ;
    else
    { /* backup in the input */
        ungetNextChar();
        save = FALSE;
        currentToken = ASSIGN;
    }
    break;

```

## result

---

- test.1.txt

```
base
> ./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
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
```

- test.3.txt (i generate test case)

```
test.3.txt
> ./cminus_cimpl test.3.txt
C-MINUS COMPILATION: test.3.txt
1: /
1: *
2: /
2: /
3: ==
3: =
4: ID, name= elseif
5: ID, name= elseif
6: ID, name= elseif
7: ID, name= v1ovoid
8: ID, name= until
11: *
12: EOF
```

```
test.3.txt
/*
//
===
elseif
elseif
elseif
v1ovoid
until
/****1/
while11
****/*
```

## #2 cminus\_lex

- cminus\_lex 은 cminus 의 문법에 맞는 DFA를 찾고, cminus.1 의 rule part를 수정해서 구현한다.
- Definition Section

- 우선 `identifier` 은 알파벳으로 시작하고 그 이후엔 숫자 알파벳 조합 어느것이든 가능하므로 정규식 표현으로 `identifier {letter}({letter}|{digit})*`

- Rule Section

- `cminus_cimpl` 과 동일하게 reserved words로 사용되지 않는 `THEN`, `END`, `REPEAT`, `UNTIL`, `READ`, `WRITE` 를 지우고 `WHILE`, `RETURN`, `INT`, `VOID` 를 추가했다.
- `"=="` 은 `EQ`, `'='` 은 `ASSIGN` 을 return하도록 했고, special symbols로 추가적으로 사용되는 `LE`, `GT`, `GE`, `NE`, `LBACE`, `RBRACE`, `LCURLY`, `RCULRY`, `COMMA` 를 추가하고 각각의 symbol에 맞는 것을 return 하도록 했다.
- 중점적으로 다룬 부분은 Comment 부분이다.

- Comment

- DFA는 이전의 `cminus_cimpl` 과 동일하다.
- `"/*"` 이 들어올 경우 `"*/"` 이 다시 나오면 주석이 끝나기 때문에 `"*/"` 을 찾을 때까지 do-while loop를 돌도록 했다.
- 우선 처음 `'*'` 을 찾을 때까지 token을 읽는다. 도중에 `EOF` 를 만나면 break하고 `newline` 을 만나면 `lineno` 을 올려준다.
- `'*'` 을 찾으면 바로 이어서 `'/'` 가 나오는지 찾는다. 바로 이어서 `'/'` 가 나오지 않을 경우 다시 `'*'` 을 찾는 과정을 반복한다. 이때도 도중에 `EOF` 를 만나면 break하고 `newline` 을 만나면 `lineno` 을 올려준다.
- 코드는 다음과 같다.

```

"/*"          { char c;
                int check = 0;
                do
                { c = input();
                  if (c == EOF || c == '\0' || (c == '/' && check))
break;

                  if (c == '\n') lineno++;
                  check = (c == '*');
                } while (1);
                }
.              {return ERROR;}

```

- comment가 다 닫히지 않은채로 `EOF` 를 만나게 됐을 때 `EOF` 를 인식하지 못하는 경우가 생겨서 `'\0'` 을 읽었을 때 도 종료하도록 조건을 추가했다.

## result

o test.1.txt

```
> ./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: {
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
```

o test.3.txt and test.4.txt (i generate test case)

test.4.txt

```
void main(void)
{
    int i; int x[5];

    i = 0;
}
```

test.3.txt

```
/*
//
===
elseif
elseelse
elseelse
vlovvoid
until
/****1/
while11
*/**/*
```

```
> ./cminus_lex test.4.txt
C-MINUS COMPILATION: test.4.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: EOF
```

```
> ./cminus_lex test.3.txt
C-MINUS COMPILATION: test.3.txt
1: /
1: *
2: /
2: /
3: ==
3: =
4: ID, name= elseif
5: ID, name= elseelse
6: ID, name= elseelse
7: ID, name= vlovvoid
8: ID, name= until
11: *
11: EOF
```