C-Minus Scanner

Environment : Ubuntu 18.04

Cimpilation method : c언어 file과 lex를 위한 .l 파일을 수정하고 Makefile을 이용해 gcc로 한번에 Compile 했다.

1) C code

Scan.c

```
typedef enum
   { START,INEQ,INLT,INGT,INNE,INOVER,INCOMMENT,INCOMMENT_,INNUM,INID,DONE }
   StateType;
```

DFA를 위한 State Type에 INEQ, INLT, INGT, INNE, INOVER, INCOMMENT, INCOMMENT_를 추가해준다.

```
/* lookup table of reserved words */
static struct
    { char* str;
      TokenType tok;
    } reservedWords[MAXRESERVED] = {
      {"if",IF},
      {"else",ELSE},
      {"while",WHILE},
      {"return",RETURN},
      {"int",INT},
      {"void",VOID},
    };
```

Reserved Word를 위한 lookup table을 변경한 Reserved Word로 바꾸어준다.

```
else if (c == '=')
{
  save = FALSE;
  state = INEQ;
}
else if (c == '/')
{
  save = FALSE;
  state = INOVER;
}
else if (c == '<')
{
  save = FALSE;
  state = INLT;
}
else if (c == '>')
{
  save = FALSE;
  state = INGT;
}
else if (c == '!')
{
  save = FALSE;
  state = INNE;
}
```

=, /, <, >, ! symbol을 읽었을 때 각각 ASSIGN/EQ, OVER/COMMENT, LT/LE, GT/GE, NE인지 확인하기 위해 INEQ, INOVER, INLT, INGT, INNE state로 이동한다.

```
case '*':
  currentToken = TIMES;
  break;
case '(':
  currentToken = LPAREN;
  break;
case ')':
  currentToken = RPAREN;
  break;
case '[':
  currentToken = LBRACE;
  break;
case ']':
  currentToken = RBRACE;
  break;
case '{':
  currentToken = LCURLY;
  break;
case '}':
  currentToken = RCURLY;
  break;
case ';':
  currentToken = SEMI;
  break;
case ',':
  currentToken = COMMA;
  break;
```

그 밖에 추가한 symbol들에 대해서 각각의 TokenType으로 설정되도록 추가해준다.

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

INOVER state인 경우(/을 읽었을 때)
바로 뒤에 *가 나온다면 COMMENT가 시작되므로 INCOMMENT로 state를 움직여준다.

만약 다른 char가 나온다면 ungetNextChar로 읽은 char를 되돌리고 Token을 OVER로 설정하고 state를 DONE으로 설정한다.

```
case INCOMMENT:
  save = FALSE;
  if ( c == '*')
  {
    state = INCOMMENT_;
  }
  else if ( c == EOF)
  {
    state = DONE;
    currentToken = ENDFILE;
  }
  break;
```

INCOMMENT state인 경우
COMMENT가 시작되고 중간에 모든 단어들은 INCOMMENT 상태이다.
이때 만약 *을 읽었다면 */로 COMMENT를 마무리하는 단계인지 확인하기 위해 INCOMMENT_ state로 이동한다.

만약 COMMENT 도중에 파일을 다 읽게 되었다면 모든 token을 다 읽었다고 표시하는 ENDFILE로 설정하고 state를 DONE으로 설정한다.

```
case INCOMMENT_:
  save = FALSE;
  if ( c =='/')
    state = START;
  else if ( c == '*')
    state = INCOMMENT_;
  else if ( c == EOF )
  {
    state= DONE;
    currentToken = ENDFILE;
  }
  else
    state = INCOMMENT;
  break;
```

INCOMMENT_ state인 경우
/를 읽었다면 */로 COMMENT가 마무리 되므로 state를 다시 START로 변경해주어 새로운 Token을 읽을 준비를 한다.
*를 읽었다면
**/ 같은 경우인지 확인하기 위해 다시 INCOMMENT_ state로 이동한다.
EOF인 경우는 파일을 다 읽게 되었으므로 모든 token을 다 읽었다고 표시하는 ENDFILE로 설정하고 state를 DONE으로 설정한다.
그 밖에 경우는 COMMENT가 끝나지 않았으므로 INCOMMENT state로 다시 이동한다.

```
case INEQ:
  state = DONE;
  if (c == '=')
    currentToken = EQ;
  else
  {
    ungetNextChar();
    currentToken = ASSIGN;
  }
  break;
```

INEQ state인 경우
=가 나온다면 == 가 되므로 EQ state로 이동한다.

만약 다른 char가 나온다면 읽은 char를 되돌리고 ASSIGN state로 이동한다.

INNE state인 경우
다음 char가 = 인경우 != 가 되므로 NE state로 이동한다.

```
case INNE:
  state = DONE;
  if (c == '=')
    currentToken = NE;
  else
  {
    ungetNextChar();
    save = FALSE;
    currentToken = ERROR;
  }
  break;
```

그 밖에 다른 char인 경우는 다른 symbol case가 없으므로 현재 읽은 char를 지워주고 Token을 ERROR로 변경해준다.

INLT/ INGT인 경우

```
case INLT:
  state = DONE;
  if (c == '=')
    currentToken = LE;
  else
  {
    ungetNextChar();
    currentToken = LT;
  }
  break;
case INGT:
  state = DONE;
  if (c == '=')
    currentToken = GE;
  else
  {
    ungetNextChar();
    currentToken = GT;
  }
  break;
```

다음 char가 =인 경우 <= / >= 가 되므로 LE, GE state로 이동한다.

그 밖에 다른 char인 경우 읽은 char를 되돌리고 LT/GT state로 이동한다.

```
case INID:
  if (!isalpha(c) && !isdigit(c))
  { /* backup in the input */
    ungetNextChar();
    save = FALSE;
    state = DONE;
    currentToken = ID;
  }
  break;
```

INID state인 경우

기존 syntax와 다르게 ID에 숫자가 들어가는 경우가 포함되므로 INID state에서 읽은 char가 alphabet이나 digit이 아니라면 ID가 끝남을 알 수 있다. 따라서 해당 경우에 읽은 char를 되돌리고 Token을 ID로 반환한다.

Result

```
ww@ww-VirtualBox:~/loucomp$ ./cminus_cimpl test1.cm

C-MINUS COMPILATION: test1.cm
       4: reserved word: int
       4: ID, name= gcd1
       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= gcd1
15: (
15: ID, name= x
15: ,
15: ID, name= y
15: )
15: )
15: ;
16: }
17: EOF
```

2) Lex

Cminus.l

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

identifier에 문자로 시작하고 이후에 숫자가 들어갈 수 있으므로 다음과같이 변경해준다.

```
"if"            {return IF;}
"else"          {return ELSE;}
"while"         {return WHILE;}
"return"        {return RETURN;}
"int"           {return INT;}
"void"          {return VOID;}
"="             {return ASSIGN;}
"=="            {return EQ;}
"!="            {return NE;}
"<"             {return LT;}
"<="            {return LE;}
">"             {return GT;}
">="            {return GE;}
"+"             {return PLUS;}
"-"             {return MINUS;}
"*"             {return TIMES;}
"/"             {return OVER;}
"("             {return LPAREN;}
")"             {return RPAREN;}
"["             {return LBRACE;}
"]"             {return RBRACE;}
"{"             {return LCURLY;}
"}"             {return RCURLY;}
";"             {return SEMI;}
","             {return COMMA;}
```

추가된 Reserved word와 special symbol들에 대해서 해당 Token name으로 return 해준다.

```
"/*"        { char c;
              while (1)
              { c = input();
                if (c == EOF) break;
                if (c == '\n') lineno++;
                if (c == '*')
                {
                  INCOMMENT_:
                  c = input();
                  if ( c == '/') break;
                  if ( c == EOF) break;
                  if ( c == '*') goto INCOMMENT_;
                }
              }
            }
```

/*인 경우 COMMENT가 시작되므로 while문을 돌면서 */ 때까지 읽어준다.

이때 **/ 같은 경우를 확인하기 위해 INCOMMENT_ lable을 추가해주고 goto 문을 이용해서 로 이동해 COMMENT가 계속되는지 끝나는지 확인하여준다.

Result

```
ww@ww-VirtualBox:~/loucomp$ ./cminus_lex test1.cm

C-MINUS COMPILATION: test1.cm
        4: reserved word: int
        4: ID, name= gcd1
        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= gcd1
15: (
15: ID, name= x
15: ,
15: ID, name= y
15: )
15: )
15: ;
16: }
17: EOF
```

Lex와 c code 모두 똑같은 결과로 scan 함을 알 수 있다.