1. Code

- scan.c

```
switch (state)
{ case START:
    if (isdigit(c))
      state = INNUM;
    else if (isalpha(c))
      state = INID;
    else if (c == '/') {
      state = INOVER;
      save = FALSE;
    }
    else if (c == '<')
      state = INLT;
    else if (c == '>')
      state = INGT;
    else if (c == '=')
      state = INEQ;
    else if (c == '!')
      state = INNE;
```

‘/’ 에 대해 나누기 연산과 주석의 동시 상태이므로 선결정 상태의 나누기 연산으로 분기했습니다.
‘<’ 에 대해 작다와 작거나 같다의 동시 상태이므로 선결정 상태의 작다로 분기했습니다.
‘>’ 에 대해 크다와 크거나 같다의 동시 상태이므로 선결정 상태의 크다로 분기했습니다.
‘=’ 에 대해 대입과 같다의 동시 상태이므로 선결정 상태의 대입으로 분기했습니다.
‘!’ 에 대해 다음 문자를 읽어야 상태를 결정할 수 있으므로 NOT으로 분기했습니다.
나머지 토큰에 대해서는 입력 시 결정 상태이므로 토큰을 반환했습니다.

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

INOVER 상태에서 ‘*’ 입력 시 INCOMMENT 상태로 분기됩니다.
INCOMMENT 상태에서는 ‘*’ 문자가 오는지 검사해 INCOMMENT_ 상태로 분기합니다.
INCOMMENT_ 상태에서는 ‘/’ 문자가 오면 START 상태로 분기해 저장될 입력을 준비합니다.
‘*’ 가 온다면 상태 전이가 일어나지 않고 INCOMMENT_ 상태로 다시 분기합니다.
다른 문자가 온다면 다시 INCOMMENT 상태로 분기합니다.

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

‘=’ 가 오면 같다 토큰을 반환합니다.
아니라면, 읽었던 문자를 되돌리고 대입 토큰을 반환합니다.

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

‘=’ 가 오면 작거나 같다 토큰을 반환합니다.
아니라면, 읽었던 문자를 되돌리고 작다 토큰을 반환합니다.

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

‘=’ 가 오면 크거나 같다 토큰을 반환합니다.
아니라면, 읽었던 문자를 되돌리고 크다 토큰을 반환합니다.

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

‘=’ 가 오면 다르다 토큰을 반환합니다.
아니라면, 읽었던 문자를 되돌리고 ERROR를 반환합니다.

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

‘*’ 가 오면 주석이므로 INCOMMENT 상태로 분기합니다.
아니라면, 읽었던 문자를 되돌리고 나누기 토큰을 반환합니다.

- cminus.l

```
identifier  {letter}({letter}|{digit})*
```

식별자는 문자로 시작하고 이후에는 문자와 숫자로 구성됩니다.

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

다음과 같은 토큰을 추가했습니다.

```
"/*"             { char c;
                   int state = 0; // scan.c의 INCOMM
                  do
                  { c = input();
                    if (c == 0) {
                        break; // EOF
                    }
                    if (!state) {
                        /* INCOMMENT 상태 */
                        if (c == '\n') lineno++;
                        else if (c == '*') state = 1;
                    } else {
                        /* INCOMMENT_ 상태 */
                        if (c == '/') break;
                        else if (c != '*') state = 0;
                        // *가 아니라면 INCOMMENT 상태
OMMENT_ 상태를 유지합니다.
                    }
                  } while (1); // body에서 모든 경우에
었으므로 무한 반복문 처리했습니다.
                }
```

주석입니다.
‘*’ 문자 입력 시 INCOMMENT_ 상태로 전이합니다.
INCOMMENT_ 상태에서 ‘/’ 문자 입력 시 주석을 끝냅니다.

‘*’ 입력 시 상태를 유지합니다.
다른 문자 입력 시 INCOMMENT_ 상태로 다시 돌아갑니다.

2. Result

사진이 전부 들어가지 않기에 테스트 코드 모음집을 하나 만들어 스크린샷을 올립니다.

```
int testfunc(int a, int b) {
        /* This is test comments */
        /* This is test comments **/
        / * This is test comments */
        /* This is test comments * / */

        int a123b234;
        int 123a234b;
        int testarr[10];

        if (a == b) {
                a = b - 1;
                b = a / 3;
                testarr[3] = 4;
        }

        if (a > b) {
                b = a * b;
        }

        if (a < b) {
                a = a * b;
        }

        if (a >= b) {
                b = a - b + 1;
        }

        if (a <= b) {
                a = b - a + 1;
        }

        if (a != b) {
                a = b;
        }

        int i = 0;
        while (i < 10) {
                i = i + 1;
                testarr[i] = testarr[i - 1];
        }

        return a;
}
int main(void) {
        testfunc(1, 1);
        testfunc(1, 2);
        testfunc(2, 1);

        return 0;
}

/* This is test comments
```

lex도 똑같이 나왔습니다.

```
chyun@Ohui-MacBookPro 1_Scanner % ./cminus_cimpl example/test6.cm

C-MINUS COMPILATION: example/test6.cm
        1: reserved word: int
        1: ID, name= testfunc
        1: (
        1: reserved word: int
        1: ID, name= a
        1: ,
        1: reserved word: int
        1: ID, name= b
        1: )
        1: {
        4: /
        4: *
        4: ID, name= This
        4: ID, name= is
        4: ID, name= test
        4: ID, name= comments
        4: *
        4: /
        7: reserved word: int
        7: ID, name= a123b234
        7: ;
        8: reserved word: int
        8: NUM, val= 123
        8: ID, name= a234b
        8: ;
        9: reserved word: int
        9: ID, name= testarr
        9: [
        9: NUM, val= 10
        9: ]
        9: ;
        11: reserved word: if
        11: (
        11: ID, name= a
        11: ==
        11: ID, name= b
        11: )
        11: {
        12: ID, name= a
        12: =
        12: ID, name= b
        12: -
          12: NUM, val= 1
          12: ;
          13: ID, name= b
          13: =
          13: ID, name= a
          13: /
          13: NUM, val= 3
          13: ;
          14: ID, name= testarr
          14: [
          14: NUM, val= 3
          14: ]
          14: =
          14: NUM, val= 4
          14: ;
          15: }
          17: reserved word: if
          17: (
          17: ID, name= a
          17: >
          17: ID, name= b
          17: )
          17: {
          18: ID, name= b
          18: =
          18: ID, name= a
          18: *
          18: ID, name= b
          18: ;
          19: }
          21: reserved word: if
          21: (
          21: ID, name= a
          21: <
          21: ID, name= b
          21: )
          21: {
          22: ID, name= a
          22: =
          22: ID, name= a
          22: *
          22: ID, name= b
          22: ;
          23: }
          25: reserved word: if
          25: (
          25: ID, name= a
          25: >=
          25: ID, name= b
          25: )
          25: {
          26: ID, name= b
          26: =
          26: ID, name= a
          26: -
          26: ID, name= b
          26: +
          26: NUM, val= 1
          26: ;
          27: }
          29: reserved word: if
          29: (
          29: ID, name= a
          29: <=
          29: ID, name= b
          29: )
          29: {
          30: ID, name= a
          30: =
          30: ID, name= b
          30: -
          30: ID, name= a
          30: +
          30: NUM, val= 1
          30: ;
          31: }
          33: reserved word: if
          33: (
          33: ID, name= a
          33: !=
          33: ID, name= b
          33: )
          33: {
          34: ID, name= a
          34: =
          34: ID, name= b
          34: ;
          35: }
          37: reserved word: int
          37: ID, name= i
          37: =
          37: NUM, val= 0
          37: ;
            38: reserved word: while
            38: (
            38: ID, name= i
            38: <
            38: NUM, val= 10
            38: )
            38: {
            39: ID, name= i
            39: =
            39: ID, name= i
            39: +
            39: NUM, val= 1
            39: ;
            40: ID, name= testarr
            40: [
            40: ID, name= i
            40: ]
            40: =
            40: ID, name= testarr
            40: [
            40: ID, name= i
            40: -
            40: NUM, val= 1
            40: ]
            40: ;
            41: }
            43: reserved word: return
            43: ID, name= a
            43: ;
            44: }
            46: reserved word: int
            46: ID, name= main
            46: (
            46: reserved word: void
            46: )
            46: {
            47: ID, name= testfunc
            47: (
            47: NUM, val= 1
            47: ,
            47: NUM, val= 1
            47: )
            47: ;
            48: ID, name= testfunc
            48: (
            48: NUM, val= 1
            48: ,
            48: NUM, val= 2
            48: )
            48: ;
            49: ID, name= testfunc
            49: (
            49: NUM, val= 2
            49: ,
            49: NUM, val= 1
            49: )
            49: ;
            51: reserved word: return
            51: NUM, val= 0
            51: ;
            52: }
            56: EOF
```