1. Scanner

Implementation of C-Scanner using C-code & flex by cminus.l

Compilation method and environment

1. Compilation method

C-code: make clean_cminus 명령으로 기존 실행파일 제거, make cminus 명령으로 최신 실행파일 생성

```
youngseo@YS-VirtualBox:~/Desktop/2020_ELE4029_2016024784/1_Scanner$ make clean_c
rm cminus main.o util.o scan.o
youngseo@YS-VirtualBox:~/Desktop/2020_ELE4029_2016024784/1_Scanner$ make cminus
gcc    -c -o main.o main.c
gcc    -c -o util.o util.c
gcc    -c -o scan.o scan.c
gcc  main.o util.o scan.o globals.h -o cminus
```

l: make clean_cminus_flex 명령으로 기존 실행파일 제거, make cminus_flex 명령으로 최신 실행파일 생성

```
youngseo@YS-VirtualBox:~/Desktop/2020_ELE4029_2016024784/1_Scanner$ make clean_cminus_flex
rm cminus_flex main.o util.o lex.yy.o
youngseo@YS-VirtualBox:~/Desktop/2020_ELE4029_2016024784/1_Scanner$ make cminus_flex
gcc    -c -o main.o main.c
gcc    -c -o util.o util.c
flex cminus.l
gcc  -c lex.yy.c -lfl
gcc  main.o util.o lex.yy.o -o cminus_flex -lfl
```

2. Environment: Ubuntu 20.04.1 LTS

Explanation about how to implement and how to operate

globals.h

```
25 /* MAXRESERVED = the number of reserved words */
26 #define MAXRESERVED 6
27
28 typedef enum
29     /* book-keeping tokens */
30    {ENDFILE,ERROR,
31     /* reserved words */
32     IF, ELSE, WHILE, RETURN, INT, VOID, /* discarded THEN, END, REPEAT, UNTIL, READ, WRITE, */
33     /* multicharacter tokens */
34     ID,NUM,
35     /* special symbols */
36     ASSIGN, EQ, NE, LT, LE, GT, GE, PLUS, MINUS, TIMES, OVER, LPAREN, RPAREN, LBRACE, RBRACE, LCURLY, RCURLY, SEMI,
   COMMA
37    } TokenType;
```

MAXRESERVED 6으로 변경 - IF, ELSE, WHILE, RETURN, INT, VOID

Tiny 토큰들을 C Minus 토큰으로 변경

main.c

```
11 #define NO_PARSE TRUE
```

```
39 /* allocate and set tracing flags */
40 int EchoSource = TRUE;
41 int TraceScan = TRUE;
42 int TraceParse = FALSE;
43 int TraceAnalyze = FALSE;
44 int TraceCode = FALSE;
```

Flag 수정

util.c

```
26     case ASSIGN: fprintf(listing,"=\n"); break;
27     case EQ: fprintf(listing,"==\n"); break;
28     case NE: fprintf(listing,"!=\n"); break;
29     case LT: fprintf(listing,"<\n"); break;
30     case LE: fprintf(listing,"<=\n"); break;
31     case GT: fprintf(listing,">\n"); break;
32     case GE: fprintf(listing,">=\n"); break;
33     case LPAREN: fprintf(listing,"(\n"); break;
34     case RPAREN: fprintf(listing,")\n"); break;
35     case LBRACE: fprintf(listing,"[\n"); break;
```

```
36     case RBRACE: fprintf(listing,"]\n"); break;
37     case LCURLY: fprintf(listing,"{\n"); break;
38     case RCURLY: fprintf(listing,"}\n"); break;
39     case SEMI: fprintf(listing,";\n"); break;
40     case COMMA: fprintf(listing,",\n"); break;
41     case PLUS: fprintf(listing,"+\n"); break;
42     case MINUS: fprintf(listing,"-\n"); break;
43     case TIMES: fprintf(listing,"*\n"); break;
44     case OVER: fprintf(listing,"/\n"); break;
45     case ENDFILE: fprintf(listing,"EOF\n"); break;
```

C Minus token들을 print하기 위한 case 추가

scan.c

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

C Minus DFA를 위한 state로 변경

```
/* 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},
       /* discarded  {"then",THEN},{"end",END},{"repeat", REPEAT},{"until",UNTIL},{"read",READ},{"write",WRITE} */
   };
```

C Minus를 위한 reserved word로 변경

```
    // switch start
    switch (state)
    { case START:
        if (isdigit(c))
          state = INNUM;
        else if (isalpha(c))
          state = INID;
        else if ((c == ' ') || (c == '\t') || (c == '\n'))
          save = FALSE;
        else if (c == '=')
          state = INEQ;
        else if (c == '!')
          state = INNE;
        else if (c == '<')
          state = INLT;
        else if (c == '>')
          state = INGT;
        else if (c == '/')
        { /* Do not save comment */
          save = FALSE;
          state = INOVER;
        }
```

```
        else
        {
          state = DONE;
          switch (c)
          { case EOF:
              save = FALSE;
              currentToken = ENDFILE;
              break;
            case '+':
              currentToken = PLUS;
              break;
            case '-':
              currentToken = MINUS;
              break;
            case '*':
              currentToken = TIMES;
              break;
            case '(':
              currentToken = LPAREN;
              break;
            case ')':
              currentToken = RPAREN;
              break;
```

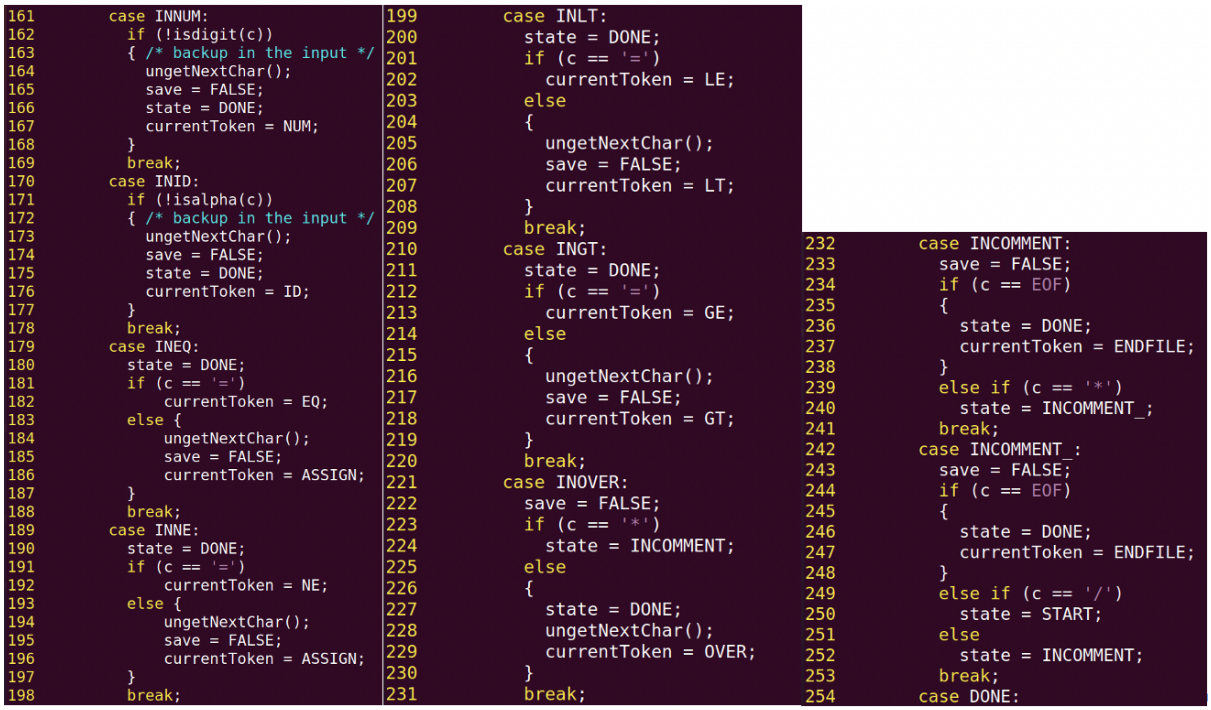
```
            case '{':
              currentToken = LCURLY;
              break;
            case '}':
              currentToken = RCURLY;
              break;
            case '[':
              currentToken = LBRACE;
              break;
            case ']':
              currentToken = RBRACE;
              break;
            case ';':
              currentToken = SEMI;
              break;
            case ',':
              currentToken = COMMA;
              break;
            default:
              currentToken = ERROR;
              break;
          }
        }
        break;
```

C Minus를 위한 DFA 수정

최초 START state로 DFA를 시작하는 경우 토큰이 한 글자인 것과 한 글자 이상인 것으로 구분.

한 글자 토큰은 state를 DONE으로 변경하여 더 이상 문자를 읽지 않고 자신을 tokenString에 저장하며, currentToken을 자신의 type으로 변경한다.

한 글자 이상의 토큰은 최초의 문자에 따라 나올 수 있는 token이 달라지기 때문에 그에 맞게 state를 갖도록 변경. 예를 들어 '='의 경우 INEQ로 state를 변경하여, '=' 또는 '=='에 대비한다. 즉, 현재에 token을 결정하는 것이 아닌 추후 등장하는 문자를 본 뒤 판단한다.



```
161         case INNUM:
162           if (!isdigit(c))
163           { /* backup in the input */
164             ungetNextChar();
165             save = FALSE;
166             state = DONE;
167             currentToken = NUM;
168           }
169           break;
170         case INID:
171           if (!isalpha(c))
172           { /* backup in the input */
173             ungetNextChar();
174             save = FALSE;
175             state = DONE;
176             currentToken = ID;
177           }
178           break;
179         case INEQ:
180           state = DONE;
181           if (c == '=')
182               currentToken = EQ;
183           else {
184               ungetNextChar();
185               save = FALSE;
186               currentToken = ASSIGN;
187           }
188           break;
189         case INNE:
190           state = DONE;
191           if (c == '=')
192               currentToken = NE;
193           else {
194               ungetNextChar();
195               save = FALSE;
196               currentToken = ASSIGN;
197           }
198           break;
```

```
199         case INLT:
200           state = DONE;
201           if (c == '=')
202             currentToken = LE;
203           else
204           {
205             ungetNextChar();
206             save = FALSE;
207             currentToken = LT;
208           }
209           break;
210         case INGT:
211           state = DONE;
212           if (c == '=')
213             currentToken = GE;
214           else
215           {
216             ungetNextChar();
217             save = FALSE;
218             currentToken = GT;
219           }
220           break;
221         case INOVER:
222           save = FALSE;
223           if (c == '*')
224             state = INCOMMENT;
225           else
226           {
227             state = DONE;
228             ungetNextChar();
229             currentToken = OVER;
230           }
231           break;
```

```
232         case INCOMMENT:
233           save = FALSE;
234           if (c == EOF)
235           {
236             state = DONE;
237             currentToken = ENDFILE;
238           }
239           else if (c == '*')
240             state = INCOMMENT_;
241           break;
242         case INCOMMENT_:
243           save = FALSE;
244           if (c == EOF)
245           {
246             state = DONE;
247             currentToken = ENDFILE;
248           }
249           else if (c == '/')
250             state = START;
251           else
252             state = INCOMMENT;
253           break;
254         case DONE:
```

state가 각각의 최초 문자에 맞게 변경된 token들은 while 문을 다시 돌아 위 코드에 도달하게 된다.

INNUM – 최초 문자가 숫자인 경우. DFA는 숫자가 아닌 문자가 나올 때까지 계속 while 문을 반복하며 문자를 tokenString에 저장한다. 숫자가 아닌 문자가 나오면 해당 문자를 다음 while문에서 한번 더 확인할 수 있게 ungetNextChar()를 호출한다. 또한 save를 FASLE로 변경하여 숫자가 아닌 문자를 tokenString에 저장하지 않고, state를 DONE으로, currentToken을 NUM으로 변경한다.

INID – 최초 문자가 letter인 경우이다. DFA는 **INNUM**의 경우와 동일한 방법으로 동작한다.

INEQ – 최초 문자가 '='인 경우이다. 두번째 문자가 '='이면 currentToken은 EQ가 된다. '='가 아닌 경우 currentToken은 ASSIGN이 되고 현재 읽은 문자를 다시 판단하기 위해 ungetNextChar()를 호출.

INNE, **INLT**, **INGT** – **INEQ**와 동일한 방법으로 동작한다.

INOVER – 최초 문자가 '/'인 경우이다. 두번째 문자가 '*'인 경우 comment이므로 state를 **INCOMMENT**로 변경한다. 아니라면 over을 의미함으로 state를 DONE으로, currentToken을 OVER로 변경한다.

INCOMMENT – EOF가 나온다면 파일을 다 읽은 것이므로 state를 DONE으로, currentToken을 ENDFILE로 변경한다. '*'의 경우 comment의 종료를 기대할 수 있기 때문에 state를 **INCOMMENT_**로 변경한다. 이외의 경우는 전부 comment를 의미하므로 아무것도 하지 않는다.

INCOMMENT_ – EOF가 나온다면 파일을 다 읽은 것이므로 state를 DONE으로, currentToken을 ENDFILE로 변경한다. '/'가 나온 경우 '*/'이 연속적으로 나온 것이므로 comment의 종료를 나타낸다. 따라서 state를 **START**로 변경한다. 이외의 경우 이전의 문자 '*'는 그저 comment인 것이므로 state를 **INCOMMENT**로 변경한다.

Example and Result Screenshot

Example

```
/* 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));
}
```

test.1.txt

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

        i = 0;
        while( i < 5 )
        {
                x[i] = input();

                i = i + 1;
        }

        i = 0;
        while( i <= 4 )
        {
                if( x[i] != 0 )
                {
                        output(x[i]);
                }
        }
}
```

test.2.txt

C-code Result

```
C-MINUS COMPILATION: test.1.txt
   1: /* A program to perform Euclid's
   2:    Algorithm to computer gcd */
   3:
   4: int gcd (int u, int v)
        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: {
        5: {
   6:   if (v == 0) return u;
        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:   else return gcd(v,u-u/v*v);
        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: ;
   8:   /* u-u/v*v == u mod v */
   9: }
        9: }
```

```
10:
11: void main(void)
      11: reserved word: void
      11: ID, name= main
      11: (
      11: reserved word: void
      11: )
12: {
      12: {
13:   int x; int y;
      13: reserved word: int
      13: ID, name= x
      13: ;
      13: reserved word: int
      13: ID, name= y
      13: ;
14:   x = input(); y = input();
      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:   output(gcd(x,y));
      15: ID, name= output
      15: (
      15: ID, name= gcd
      15: (
      15: ID, name= x
      15: ,
      15: ID, name= y
      15: )
      15: )
      15: ;
16: }
      16: }
      17: EOF
```

Flex by cminus.l Result

```
C-MINUS COMPILATION: test.2.txt
   1: void main(void)
        1: reserved word: void
        1: ID, name= main
        1: (
        1: reserved word: void
        1: )
   2: {
        2: {
   3:    int i; int x[5];
        3: reserved word: int
        3: ID, name= i
        3: ;
        3: reserved word: int
        3: ID, name= x
        3: [
        3: NUM, val= 5
        3: ]
        3: ;
   4:
   5:    i = 0;
        5: ID, name= i
        5: =
        5: NUM, val= 0
        5: ;
   6:    while( i < 5 )
        6: reserved word: while
        6: (
        6: ID, name= i
        6: <
        6: NUM, val= 5
        6: )
   7:    {
        7: {
   8:             x[i] = input();
        8: ID, name= x
        8: [
        8: ID, name= i
        8: ]
        8: =
        8: ID, name= input
        8: (
        8: )
        8: ;
   9:
```

```
10:             i = i + 1;
      10: ID, name= i
      10: =
      10: ID, name= i
      10: +
      10: NUM, val= 1
      10: ;
11:    }
      11: }
12:
13:    i = 0;
      13: ID, name= i
      13: =
      13: NUM, val= 0
      13: ;
14:    while( i <= 4 )
      14: reserved word: while
      14: (
      14: ID, name= i
      14: <=
      14: NUM, val= 4
      14: )
15:    {
      15: {
16:             if( x[i] != 0 )
      16: reserved word: if
      16: (
      16: ID, name= x
      16: [
      16: ID, name= i
      16: ]
      16: !=
      16: NUM, val= 0
      16: )
17:             {
      17: {
18:                     output(x[i]);
      18: ID, name= output
      18: (
      18: ID, name= x
      18: [
      18: ID, name= i
      18: ]
      18: )
      18: ;
19:             }
      19: }
20:    }
      20: }
21: }
      21: }
      22: EOF
```