컴파일러설계 프로젝트1 (Scanner) 보고서

2019019016 서시언

1. Compilation method and environment

- Ubuntu 20.04.3 LTS

2. Explanation about how to implement and how to operate

- 제공된 Makefile 사용

[Method 1 (C code)]

- main.c
  - NO_PARSE, TraceScan을 TRUE로 설정한다.

```
/* set NO_PARSE to TRUE to get a scanner-only compiler */
#define NO_PARSE TRUE
/* set NO_ANALYZE to TRUE to get a parser-only compiler */
#define NO_ANALYZE FALSE

/* set NO_CODE to TRUE to get a compiler that does not
 * generate code
 */
#define NO_CODE FALSE

#include "util.h"
#if NO_PARSE
#include "scan.h"
#else
#include "parse.h"
#if !NO_ANALYZE
#include "analyze.h"
#if !NO_CODE
#include "cgen.h"
#endif
#endif
#endif

/* allocate global variables */
int lineno = 0;
FILE * source;
FILE * listing;
FILE * code;

/* allocate and set tracing flags */
int EchoSource = FALSE;
int TraceScan = TRUE;
int TraceParse = FALSE;
int TraceAnalyze = FALSE;
int TraceCode = FALSE;
```

  - 예시 출력(result.1.txt, result.2.txt)에 맞게 내용을 수정해준다.

```
  }
  listing = stdout; /* send listing to screen */
  fprintf(listing,"\nC-MINUS COMPILATION: %s\n",pgm);
```

- global.h
  - Tiny's Token(THEN, REPEAT, UNTIL, WRITE, READ, END)을 지우고, C-Minus token을 추가해주고, MAXRESERVED를 6으로 수정한다.

```
/* MAXRESERVED = the number of reserved words */
#define MAXRESERVED 6

typedef enum
    /* book-keeping tokens */
   {ENDFILE,ERROR,
    /* reserved words */
    IF,ELSE,WHILE,RETURN,INT,VOID,
    /* multicharacter tokens */
    ID,NUM,
    /* special symbols */
    ASSIGN,EQ,NE,LT,LE,GT,GE,PLUS,MINUS,TIMES,OVER,LPAREN,RPAREN,LBRACE,RBRACE,LCURLY,RCURLY,SEMI,COMMA
   } TokenType;
```

- util.c
  - 변경한 토큰들을 반영해 printToken()을 수정한다.
    - sepecial symbol은 아래와 같다.

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

```
void printToken( TokenType token, const char* tokenString )
{ switch (token)
  { case IF:
    case ELSE:
    case WHILE:
    case RETURN:
    case INT:
    case VOID:
      fprintf(listing,
         "reserved word: %s\n",tokenString);
      break;
    case ASSIGN: fprintf(listing,"=\n"); break;
    case LT: fprintf(listing,"<\n"); break;
    case NE: fprintf(listing,"!=\n"); break;
    case EQ: fprintf(listing,"==\n"); break;
    case LE: fprintf(listing,"<=\n"); break;
    case GT: fprintf(listing,">\n"); break;
    case GE: fprintf(listing,">=\n"); break;
    case LPAREN: fprintf(listing,"(\n"); break;
    case RPAREN: fprintf(listing,")\n"); break;
    case LBRACE: fprintf(listing,"[\n"); break;
    case RBRACE: fprintf(listing,"]\n"); break;
    case LCURLY: fprintf(listing,"{\n"); break;
    case RCURLY: fprintf(listing,"}\n"); break;
    case SEMI: fprintf(listing,";\n"); break;
    case COMMA: fprintf(listing,",\n"); break;
    case PLUS: fprintf(listing,"+\n"); break;
    case MINUS: fprintf(listing,"-\n"); break;
    case TIMES: fprintf(listing,"*\n"); break;
    case OVER: fprintf(listing,"/\n"); break;
    case ENDFILE: fprintf(listing,"EOF\n"); break;
    case NUM:
      fprintf(listing,
         "NUM, val= %s\n",tokenString);
      break;
    case ID:
      fprintf(listing,
         "ID, name= %s\n",tokenString);
      break;
    case ERROR:
      fprintf(listing,
         "ERROR: %s\n",tokenString);
```

- scan.c
  - state를 추가해준다.

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

  - reserved word를 수정한다.

```
/* 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},
     };
     // delete THEN, END, REPEAT, UNTIL, READ, WRITE
     // add WHILE, RETURN, INT, VOID
```

  - getToken()을 수정해준다.

```
TokenType getToken(void)
{  /* index for storing into tokenString */
   int tokenStringIndex = 0;
   /* holds current token to be returned */
   TokenType currentToken;
   /* current state - always begins at START */
   StateType state = START;
   /* flag to indicate save to tokenString */
   int save;
   while (state != DONE)
   { int c = getNextChar();
     save = TRUE;
     switch (state)
     { case START:
         if (isdigit(c))
           state = INNUM;
         else if (isalpha(c))
           state = INID;
         else if (c == '=')
           // to check if = or ==
           state = INEQ;
         else if (c == '!')
           // to check if ! or !=
           state = INNE;
         else if (c == '<')
           // to check if < or <=
           state = INLT;
         else if (c == '>')
           // to check if > or >=
           state = INGT;
         else if (c == '/')
         {
           // Since it can be COMMENT(/*), don't save
           save = FALSE;
           // to check if / or /*
           state = INOVER;
         }
```

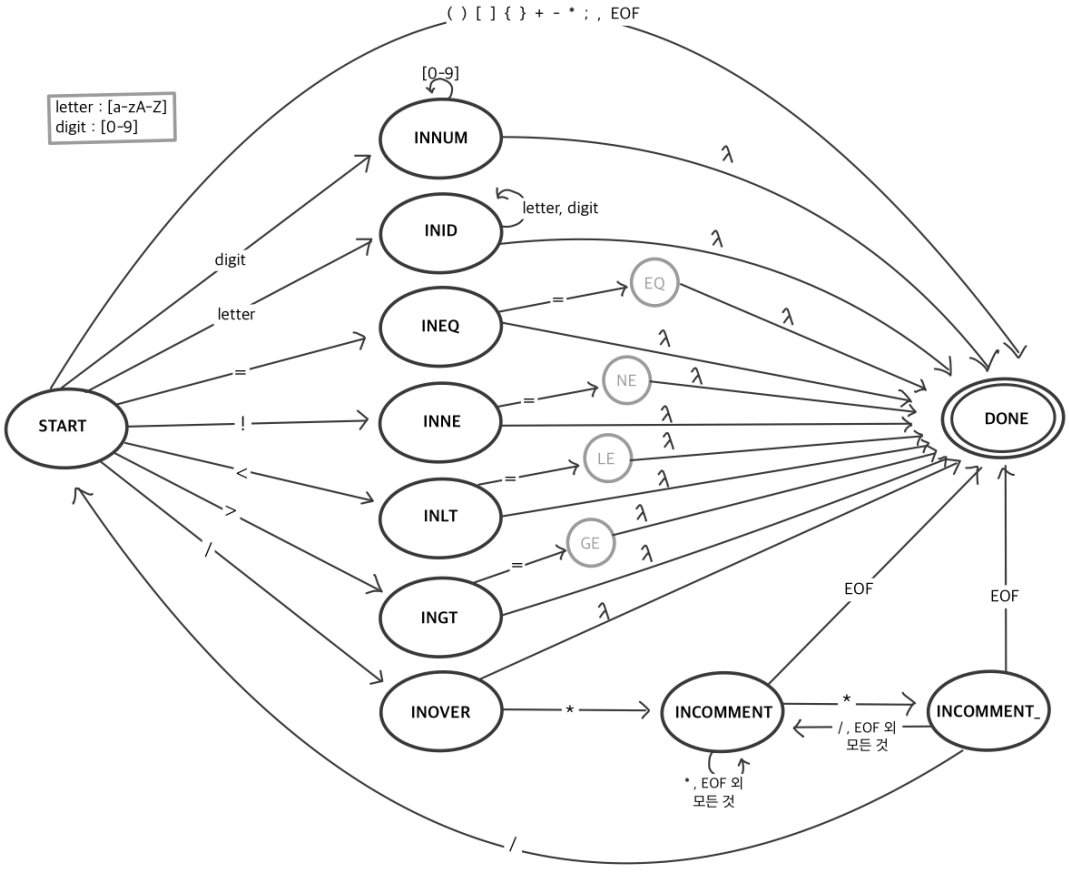
```
120         else if ((c == ' ') || (c == '\t') || (c == '\n'))
121           save = FALSE;
122         else
123         { state = DONE;
124           switch (c)
125           { case EOF:
126               save = FALSE;
127               currentToken = ENDFILE;
128               break;
129             case '+':
130               currentToken = PLUS;
131               break;
132             case '-':
133               currentToken = MINUS;
134               break;
135             case '*':
136               currentToken = TIMES;
137               break;
138             case '(':
139               currentToken = LPAREN;
140               break;
141             case ')':
142               currentToken = RPAREN;
143               break;
144             case '[':
145               currentToken = LBRACE;
146               break;
147             case ']':
148               currentToken = RBRACE;
149               break;
150             case '{':
151               currentToken = LCURLY;
152               break;
153             case '}':
154               currentToken = RCURLY;
155               break;
156             case ';':
157               currentToken = SEMI;
158               break;
159             case ',':
160               currentToken = COMMA;
161               break;
162             default:
163               currentToken = ERROR;
164               break;
165           }
166         }
167         break;
168       case INOVER:
169         if (c == '*')
170         { // COMMENT (/*)
171           save = FALSE;
172           state = INCOMMENT;
173         }
174         else
175         { // OVER (/)
176           ungetNextChar();
177           state = DONE;
178           currentToken = OVER;
179         }
180         break;
181       case INCOMMENT:
182         save = FALSE;
183         if (c == EOF)
184         { state = DONE;
185           currentToken = ENDFILE;
186         }
187         // to check COMMENT is end
188         if (c == '*') state = INCOMMENT_;
189         break;
190       case INCOMMENT_:
191         save = FALSE;
192         if (c == EOF)
193         { state = DONE;
194           currentToken = ENDFILE;
195         }
196         // end of COMMENT (*/)
197         else if (c == '/') state = START;
198         // still in COMMENT
199         else state = INCOMMENT;
200         break;
```

- 주석일 때는 save를 FALSE로 설정한다. (state가 INCOMMENT, INCOMMENT_일 때/ state가 INOVER이면서 c가 *일 때)

```
201       case INEQ:
202         state = DONE;
203         // EQ (==)
204         if (c == '=') currentToken = EQ;
205         else
206         { // ASSIGN (=)
207           ungetNextChar();
208           currentToken = ASSIGN;
209         }
210         break;
211       case INNE:
212         state = DONE;
213         // NE (!=)
214         if (c == '=') currentToken = NE;
215         else
216         { // ERROR (!)
217           ungetNextChar();
218           save = FALSE;
219           currentToken = ERROR;
220         }
221         break;
```

```
222         case INLT:
223           state = DONE;
224           // LE (<=)
225           if (c == '=') currentToken = LE;
226           else
227           { // LT (<)
228             ungetNextChar();
229             currentToken = LT;
230           }
231           break;
232         case INGT:
233           state = DONE;
234           // GE (>=)
235           if (c == '=') currentToken = GE;
236           else
237           { // GT (>)
238             ungetNextChar();
239             currentToken = GT;
240           }
241           break;
242         case INNUM:
243           if (!isdigit(c))
244           { /* backup in the input */
245             ungetNextChar();
246             save = FALSE;
247             state = DONE;
248             currentToken = NUM;
249           }
250           break;
251         case INID:
252           if (!isalpha(c) && !isdigit(c))
253           { /* backup in the input */
254             ungetNextChar();
255             save = FALSE;
256             state = DONE;
257             currentToken = ID;
258           }
259           break;
260         case DONE:
261         default: /* should never happen */
262           fprintf(listing,"Scanner Bug: state= %d\n",state);
263           state = DONE;
264           currentToken = ERROR;
265           break;
266       }
```

- getToken()이 나타내는 DFA는 다음과 같다.



- EQ, NE, LE, GE는 State가 아닌 currentToken이 EQ, NE, LE, GE임을 의미함.

- 실행

```
make
./cminus_cimpl [input_file]
```

[Method 2 (Lex(flex))]

- globals.h, main.c, util.c는 Method 1과 동일
- requirement

  sudo apt-get install flex
- cminus.l
  - tiny.l을 베이스로 함.
  - 조건에 맞게 identifier를 수정한다.

```
열기(O)                                   cminus.l
                                          ~/2021_ele4029_2019019016/1_Scanner
 8
 9 digit       [0-9]
10 number      {digit}+
11 letter      [a-zA-Z]
12 /* identifier start with letter, and then letter or digit */
13 identifier  {letter}({letter}|{digit})*
14 newline     \n
15 whitespace  [ \t]+
```

  - Token rule 수정
    - reserved word 및 special symbol을 조건에 맞게 추가해준다.

```
19 "if"             {return IF;}
20 "else"           {return ELSE;}
21 "while"          {return WHILE;}
22 "return"         {return RETURN;}
23 "int"            {return INT;}
24 "void"           {return VOID;}
25 "="              {return ASSIGN;}
26 "=="             {return EQ;}
27 "!="             {return NE;}
28 "<"              {return LT;}
29 "<="             {return LE;}
30 ">"              {return GT;}
31 ">="             {return GE;}
32 "+"              {return PLUS;}
33 "-"              {return MINUS;}
34 "*"              {return TIMES;}
35 "/"              {return OVER;}
36 "("              {return LPAREN;}
37 ")"              {return RPAREN;}
38 "["              {return LBRACE;}
39 "]"              {return RBRACE;}
40 "{"              {return LCURLY;}
41 "}"              {return RCURLY;}
42 ";"              {return SEMI;}
43 ","              {return COMMA;}
44 {number}         {return NUM;}
45 {identifier}     {return ID;}
46 {newline}        {lineno++;}
47 {whitespace}     {/* skip whitespace */}
```

    - 주석을 처리하기 위해, /* 일 때 EOF가 나오거나, 주석의 끝 (*/)이 나올 때까지 while문을 돈다.
    - while문 안에서 *이 나오면 주석의 끝인지 알기 위해 input을 하나 더 확인해본다.
      - 이때 /이면 주석의 끝이므로 break를 해준다.

```
"/*"            { char c;
                  // in COMMENT
                  do
                  { c = input();
                    if (c == '*'){
                      /* to check if end of COMMENT */
                      c = input();
                      /* if end of COMMENT, break */
                      if(c=='/') break;
                    }
                    /* to cover the case that Lex detect '\0' instead of EOF(-1) in the end of the input source. */
                    if (c == EOF || c == '\0') break;
                    if (c == '\n') lineno++;
                  } while (1);
                }
```

- 실행

  make

  ./cminus_lex [input_file]

3. Example and Result Screenshot

- test.1.txt

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

[Method 1 (C code)]

make

./cminus_cimpl test.1.txt

```
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/1_Scanner$ ./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
```

[Method 2 (Lex(Flex))]

make

./cminus_lex test.1.txt

```
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/1_Scanner$ ./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
```