Scanner Report

2019027001

문 석 원

**I. 컴파일 환경 및 방법**

**1. 컴파일 환경**

- WSL 2 (Windows Subsystem for Linux 2)

- Ubuntu 18.04.5 LTS

- flex 2.6.4

**2. 컴파일 방법**

- git clone <https://hconnect.hanyang.ac.kr/2021_ele4029_11216/2021_ele4029_2019027001.git>

- cd 2021\_ele4029\_2019027001/1\_Scanner

- make

- ./cminus\_cimpl ./[테스트\_파일명] or ./cminus\_lex ./[테스트\_파일명]

**II. 구현 및 동작 방식**

**1. globals.h**

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;

기존 Tiny의 토큰인 then, repeat, until, write, read, end를 제거하고, C-Minus에서 추가된 예약어 while, return, int, void와 special symbol NE (‘!=’), LE (‘<=’), GT (‘>’), GE (‘>=’), LBRACE (‘[‘), RBRACE (‘]’), LCURLY (‘{‘), RCURLY (‘}’), COMMA (‘,’)를 추가했다.

**2. util.c**

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 LE: fprintf(listing,"<=\n"); break;

    case GT: fprintf(listing,">\n"); break;

    case GE: fprintf(listing,">=\n"); break;

    case EQ: fprintf(listing,"==\n"); break;

    case NE: 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;

    // not changed below

  }

}

util.c의 printToken 함수도 global.h의 변경사항에 맞게 수정했다. 추가적인 차이로 ASSIGN은 기존 Tiny의 ‘:=’에서 ‘=’으로, EQ는 ‘=’에서 ‘==’로 바뀌었다는 점이 있다.

**3. scan.c**

typedef enum

   { START,INASSIGN,INNE,INLT,INGT,INOVER,INCOMMENT,INCOMMENT\_,INNUM,INID,DONE }

   StateType;

DFA의 상태는 총 11개로, 초기 상태인 START, ‘=’와 ‘==’를 식별하기 위한 상태인 INASSIGN, ‘!=’를 식별하는 INNE, ‘<’와 ‘<=’를 식별하는 INLT, ‘>’와 ‘>=’를 식별하는 INGT, ‘/’와 주석 (‘/\* \*/’)을 식별하는 INOVER, 주석을 식별하는 INCOMMENT와 INCOMMENT\_, 숫자를 식별하는 INNUM, 글자를 식별하는 INID, 그리고 종료 상태를 나타내는 DONE이 있다.

초기 상태는 START이며, 읽어들인 상태가 숫자면 INNUM 상태, 알파벳이면 INID, ‘=’이라면 INASSIGN, ‘!’이면 INNE, ‘<’이면 INLT, ‘>’이면 INGT, ‘/’이면 INOVER로 전환하고, 공백 문자나 tab, 줄바꿈 문자라면 무시하고, 나머지 글자의 경우 currentToken 변수를 해당하는 special symbol (PLUS, MINUS 등)으로 분류하고 state를 DONE으로 전환하거나, currentToken 변수를 ERROR로 설정한다.

INNUM 상태에서는 입력받은 글자가 숫자가 아니라면, token을 다 받았다고 판단하여 ungetNextChar()를 호출하고, currentToken을 NUM으로 설정하고, state를 DONE으로 바꾸고, INID 상태에서도 유사하게 입력받은 글자가 알파벳이 아니라면, token을 다 받았다고 판단하고, 동일한 절차를 수행하되, currentToken은 ID로 설정한다.

INASSIGN에서는 입력받은 글자가 ‘=’라면 token을 ‘==’ 연산자로 인식하여 currentToken을 EQ로 설정하고, 아니라면 ‘=’로 인식하여 ungetNextChar()를 호춯라고, currentToken을 ASSIGN으로 설정한다. INNE에서는 ‘=’가 등장하면 token을 ‘!=’로 인식하여 currentToken을 NE로 설정하는 반면, ‘=’가 등장하지 않았을 경우, ‘!’는 단독으로 사용되지 않으므로, state를 ERROR로 변경한다. INLT(INGT)에서는 ‘=’가 등장할 경우 token을 ‘<=’(‘>=’)로 인식하여 currentToken을 LE(GE)로 설정하고, 그렇지 않은 경우 token을 ‘<’(‘>’)로 인식하여 ungetNextChar()를 호출하고, currentToken을 LT(GT)로 변경한다.

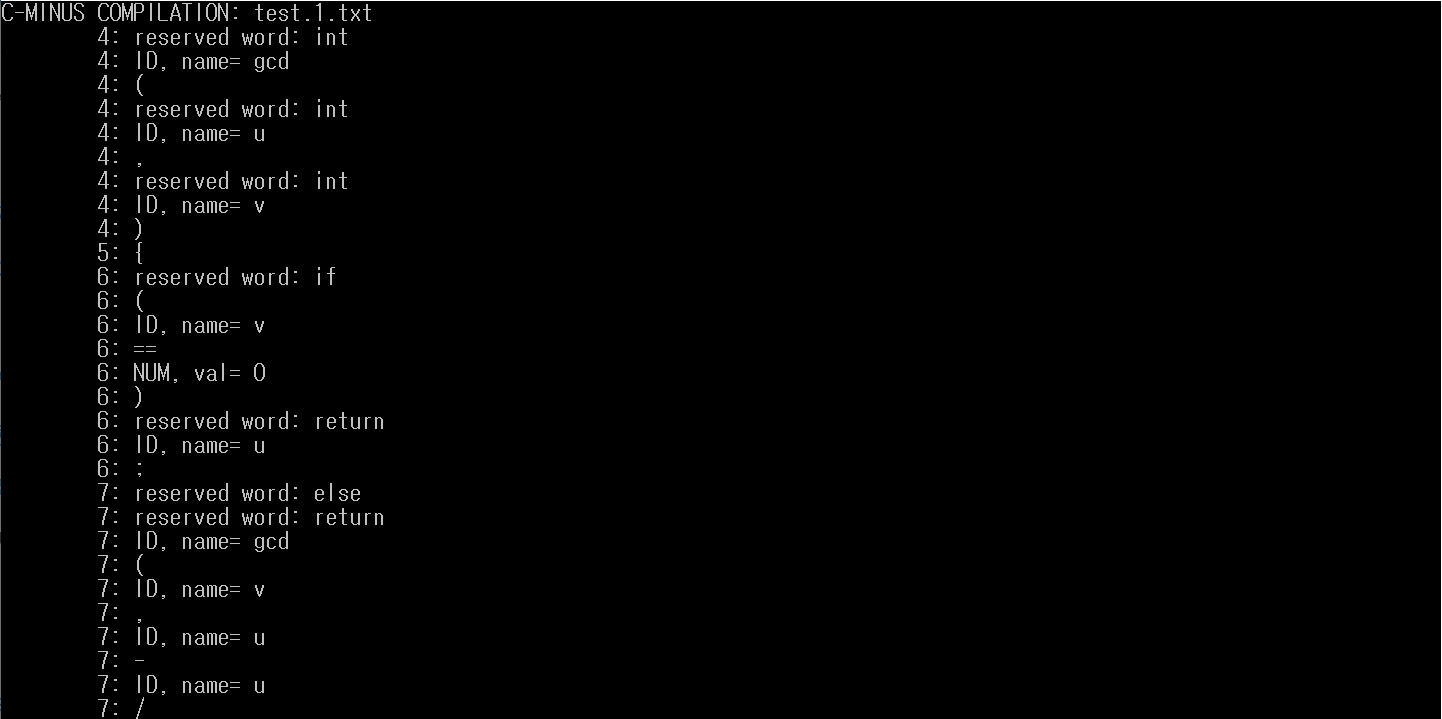
INOVER에서는 입력받은 글자가 ‘\*’이라면 여는 주석에 해당하므로, state를 INCOMMENT로 변경하고, 그 외의 경우 token을 ‘/’로 인식하여 ungetNextChar()를 호출한 후 currentToken을 OVER로 설정한다. INCOMMENT 상태에서는 입력받은 글자가 ‘\*’라면 INCOMMENT\_ 상태로 전이되며, INCOMMENT\_ 상태에서는 입력받은 글자가 ‘/’라면 주석이 끝났다고 판단하고 START state로 돌아오고, ‘\*’이라면 현재 상태에 머무르며, 그 외의 경우는 INCOMMENT 상태로 복귀한다.

**4. cminus.l**

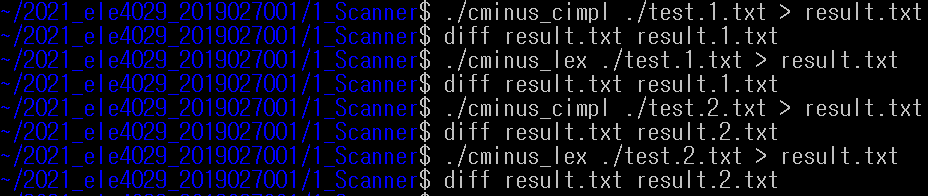
util.c와 유사하게, Tiny의 토큰들을 C-Minus 문법에 맞게 바꾸었다. 추가적인 차이로 주석이 Tiny의 ‘{’에서, C-Minus의 ‘/\*’로 변경되었다는 점이 있다.

**III. 구현 검증**

구현 검증을 위해, 제공된 테스트케이스(test.1.txt, test.2.txt)를 구현한 Scanner로 read해보았다.



./cminus\_cimpl test.1.txt를 하나, ./cminus\_lex test.1.txt를 하나 result.1.txt와 동일한 결과가 나왔으며, test.2.txt도 마찬가지로 정상 작동했다.



C-Minus Scanner로 읽은 결과를 result.txt에 담고, 이를 result.1.txt, result.2.txt와 비교해본 결과, 결과가 완전히 동일함을 알 수 있다.