Project #1 Scanner Report

2022076062 김유찬

Project Goal

C - Minus scanner를 두 가지 방법으로 구현하기

- 방법 1 : 사용자 정의 C code 이용하기
  - DFA 방법으로 토큰 인식
  - scan.c를 수정하여 lexical rules 구현
- 방법 2 : Flex를 이용하여 어휘 패턴 지정
  - Regular Expression에 의해 lexical patterns 구체화
  - cminus.l 파일 수정하여 lexical rules 정의하기

Lexical Convention - 1

- Reserved words

  int, void, if, else, while, return
- 19 symbols

  +, -, *, /, <, ≤, >, ≥, ==, ≠, =, ;, ,, (,), [,], {, }
- Identifier and number rules

  ID = letter(letter | digit)*

  NUM = digit digit*

Lexical Convention - 2

- Whitespaces
  - Spaces, newlines, tabs
  - 라인의 시작과 끝에 있는 공백은 무시하기
  - 문자, 숫자, 문자열 사이에 있는 공백은 토큰을 구분하는 데 사용하기
- Comments
  - 주석은 C 언어 표준의 형식을 따름 (/* */)
  - 한 줄 주석은 사용하지 않음 (//)
  - 주석은 중첩될 수 없음

custom C code 코드 수정

main.c

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

NO_PARSE를 TRUE로 만들어 main에서 ENDFILE이 나올 때까지 getToken을 실행하여 토큰을 얻는다.

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

TraceScan을 TRUE로 만들어 scan.c에서 각 줄마다 토큰이 출력되도록 한다.

globals.h

```c
/* 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;
```

reserved words를 IF, ELSE, WHILE, RETURN, INT, VOID 6개로 바꿔주고

special symbol을 +, -, *, /, <, <=, >, >=, ==, !=, =, ;, ,, (,), [,], {, } 이것들로 맞춘다.

util.c

globals.h에 작성한 reserved words와 symbols를 기준으로 return 값 세팅한다.

세팅한 것과 다른 기호가 나오면 ERROR가 나오도록 처리한다.

scan.c

새롭게 추가된 enum StateType

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

- INNE : '!'를 만났을 때 들어가는 상태, 이후 ERROR인지 != 인지 판단한다.
- INLT : '<'를 만났을 때 들어가는 상태, 이후 <인지 <=인지 판단한다.
- INGT : '>'를 만났을 때 들어가는 상태, 이후 >인지 >=인지 판단한다.

getToken 함수를 수정

INCOMMENT 처리

```c
else if (c == '/') // 주석 시작 감지
         {
           int next = getNextChar();
           if (next == '*'){ // '*' 이 나오면 주석임
            save = FALSE;
            state = INCOMMENT;
           }
           else{
            ungetNextChar(); // 주석이 아니면 나누기임
            currentToken = OVER;
            state = DONE;
           }
         }
```

'/'를 입력 받으면 바로 다음에 '*'가 있는 지 없는 지 판단한다. '*'가 있다면 주석 처리 상태로 넘어가고 없다면 나누기 상태로 넘어간다.

```
case INCOMMENT:
         save = FALSE;
         if (c == EOF)
         { state = DONE;
           currentToken = ENDFILE;
         }
         else if (c == '*') { // 주석이 끝나는지 판단
            if (getNextChar() == '/'){
                state = START; // 처음으로 가서 다시 토큰 받기
            }
         }
```

주석 상태에서는 주석이 끝나지 않았는데 EOF를 만나면 currentToken에 ENDFILE을 넣고 '/*' 이후 부분을 모두 주석 처리한다. '*'가 들어왔을 때 다음 문자가 '/'일때만 state를 START를 넣어 다시 토큰을 받을 준비가 되는 상태로 만든다.

INID : ID = letter(letter|digit)*을 구현하기 위해 !isdigit(c) 조건 추가

INNE : '!' 이후에 '='이 있으면 NE 처리, 아니면 ERROR처리

INLT : '<' 이후에 '='이 있으면 LE 처리, 아니면 LT처리

INGT : '>' 이후에 '='이 있으면 GE처리, 아니면 GT처리

INASSIGN : '=' 이후에 '='이 또 있으면 EQ처리, 아니면 ASSIGN처리

cminus.l

definition section

```
letter      [a-zA-Z]
identifier  {letter}({letter}|{digit})*
```

ID = ID = letter(letter|digit)*을 구현하기 위해 definition sectiond에 identifier 정의

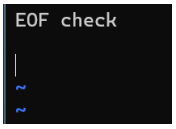
이후 rule section에 reserved words 와 symbols를 추가한다.

EOF에 대한 처리

입력에 대한 모든 것을 처리하고 입력이 없는 부분을 EOF 처리한다.

예)

EOF == 2

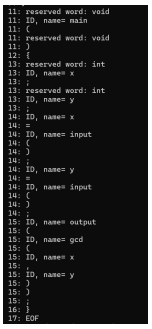


EOF == 4

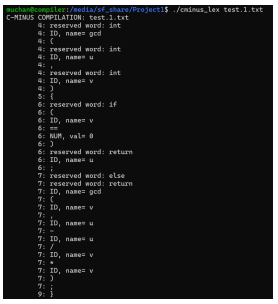
Testcase

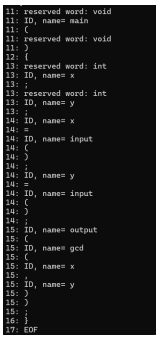
test.1.txt

./cminus_cimpl test.1.txt



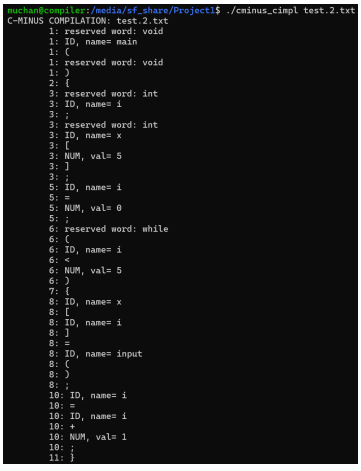
./cminus_lex test.1.txt

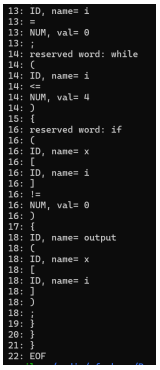




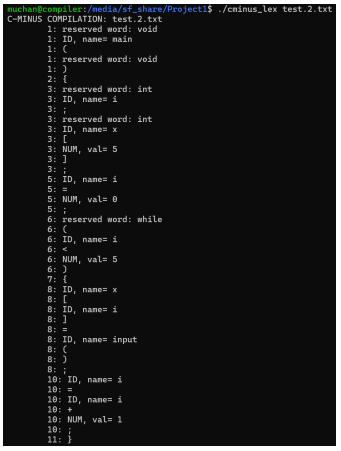
test.2.txt

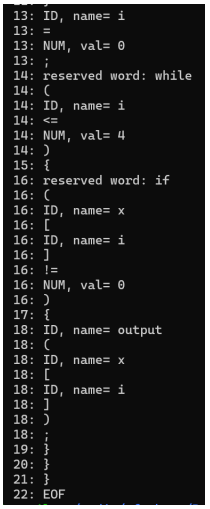
./cminus_cimpl test.2.txt





./cminus_lex test.2.txt





test.3.txt

설명

!! : !=이 아니면 !, ! 따로 에러처리

'=' 배치에 따라 분리

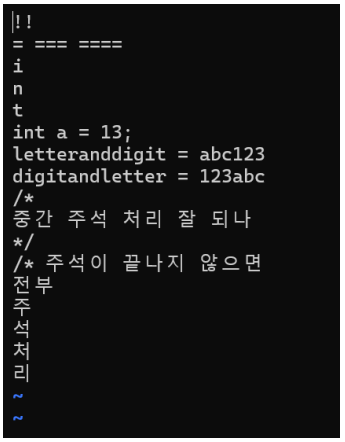
reserved word 세로 형태와 가로형태

letter + digit 과 digit + letter

/* */이 한줄에 없음

주석 끝이 끝나지 않으면 주석 처리

EOF = 18



./cminus_cimpl test.3.txt

```
muchan@compiler:/media/sf_share/Project1$ ./cminus_cimpl test.3.txt
C-MINUS COMPILATION: test.3.txt
        1: ERROR: !
        1: ERROR: !
        2: =
        2: ==
        2: =
        2: ==
        2: ==
        3: ID, name= i
        4: ID, name= n
        5: ID, name= t
        6: reserved word: int
        6: ID, name= a
        6: =
        6: NUM, val= 13
        6: ;
        7: ID, name= letteranddigit
        7: =
        7: ID, name= abc123
        8: ID, name= digitandletter
        8: =
        8: NUM, val= 123
        8: ID, name= abc
        18: EOF
```

./cminus_lex test.3.txt

```
muchan@compiler:/media/sf_share/Project1$ ./cminus_lex test.3.txt
C-MINUS COMPILATION: test.3.txt
        1: ERROR: !
        1: ERROR: !
        2: =
        2: ==
        2: =
        2: ==
        2: ==
        3: ID, name= i
        4: ID, name= n
        5: ID, name= t
        6: reserved word: int
        6: ID, name= a
        6: =
        6: NUM, val= 13
        6: ;
        7: ID, name= letteranddigit
        7: =
        7: ID, name= abc123
        8: ID, name= digitandletter
        8: =
        8: NUM, val= 123
        8: ID, name= abc
        18: EOF
```

난관

testcase를 메모장에서 작성하고 공유 폴더에 갖고와 저장을 하니 인코딩 형태가 utf-8 CRLF였다.

utf-8 CRLF는 줄바꿈이 \r\n 형태로 되어있어 왜인지는 모르겠지만 토큰 분리가 정확하지 않았다.

이후에 vim으로 작업하거나 메모장을 VScode로 실행시켜 인코딩을 utf-8 LF로 바꿔주니 줄바꿈의 형태가 \n이 되었고 이후 정상적으로 작동했다.