Project #1.Scanner

2018007938 김민관

Goal

Keyword, Symbol, Token을 정의하고 DFA를 이용하여 입력받은 문자열을 Token 단위로 Lexical Analysis한다.

환경 : Ubuntu 20.04

- **Reserved Words (Keywords)**
  - ***int void if else while return*** (lower cases)
- **Symbols**
  - **+ - * /**
  - **< <= > >= == !=**
  - **= ; ,**
  - **( ) [ ] { }**
- **Identifier and Number**
  - ***ID*** = *letter* (*letter* | *digit*)*
  - ***NUM*** = *digit digit**
  - ***letter*** = a | … | z | A | … | Z |
  - ***digit*** = 0 | … | 9

Method 1 : CMINUS_CIMPL

1. main.c

```
#define NO_PARSE TRUE

int EchoSource = FALSE;
int TraceScan = TRUE;
```

NO_PARSE와 TraceScan 을 TURE로 바꾸어주고, EchoSource의 경우 debug option을 위해 FALSE로 바꾸어 준다.

2. globals.h

```
#define MAXRESERVED 6

typedef enum
    /* book-keeping tokens */
   {ENDFILE,ERROR,
    /* reserved words */
    IF, ELSE, WHILE, RETURN, INT, VOID,  //IF,THEN,ELSE,END,REPEAT,UNTIL,READ,WRITE,
    /* 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;
```

목표하는 조건에 맞게 MAXRESERVED 는 6으로 변경해준다.

기존에 있던 then, repeat, until, write, read, end 의 symbols 를 제거해주고, token type 에 맞게 ASSIGN,EQ,NE,LT,LE,GT,GE,PLUS,MINUS,TIMES,OVER,LPAREN,RPAREN,LBRACE,RBRACE,LCURLY,RCURLY,SEMI,COM 를 추가해 준다.

3. utils.c

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

결과 물을 출력시, 다음과 같은 token이 있을 경우 print 되게끔 fprintf문을 입력한다.

4. scan.c

```
typedef enum
   { START,INCOMMENT,INNUM,INID,DONE,
   // add ==, >=, <=, !=, /* */ state
     IFEQ, IFLT, IFGT,IFNE,COMMENTTEXT, MIDCOMMENT }
   StateType;
```

==, > =, < =, ! =, /* */ 의 token을 구분하기 위해 IFEQ, IFLT, IFGT,IFNE,COMMENTTEXT, MIDCOMMENT 를 StateType enum에 추가해 준다.

```
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();

.....
.....

if ((save) && (tokenStringIndex <= MAXTOKENLEN))
       tokenString[tokenStringIndex++] = (char) c;
     if (state == DONE)
     { tokenString[tokenStringIndex] = '\0';
       if (currentToken == ID)
         currentToken = reservedLookup(tokenString);
     }
   }
   if (TraceScan) {
     fprintf(listing,"\t%d: ",lineno);
     printToken(currentToken,tokenString);
   }
   return currentToken;
}
```

TokenType getToken(void) 함수 내부의 while 문안에서 getNextChar()를 통해 문자를 하나씩 받으면서 token에 따른 명령어를 수행한다.

```
case START:
         if (isdigit(c)) state = INNUM;
         else if (isalpha(c)) state = INID;
        //  else if (c == ':')
        //    state = INASSIGN;
         else if ((c == ' ') || (c == '\t') || (c == '\n')) save = FALSE;
         else if (c == '=') state = IFEQ;
         else if (c == '<') state = IFLT;
         else if (c == '>') state = IFGT;
         else if (c == '!') state = IFNE;
         else if (c == '/')
         {
           save = FALSE;
           state = COMMENTTEXT;
         }
         else
         { state = DONE;
           switch (c)
           { case EOF:
               save = FALSE;
               currentToken = ENDFILE;
               break;
             case '=':
               currentToken = EQ;
               break;
             case '<':
               currentToken = LT;
               break;
             case '+':
               currentToken = PLUS;
               break;
             case '-':
               currentToken = MINUS;
               break;
             case '*':
               currentToken = TIMES;
               break;
             case '/':
               currentToken = OVER;
               break;
             case '(':
               currentToken = LPAREN;
               break;
             case ')':
               currentToken = RPAREN;
               break;
             case ';':
               currentToken = SEMI;
               break;
             case '{':
               currentToken = LCURLY;
               break;
             case '}':
               currentToken = RCURLY;
               break;
             case '[':
               currentToken = LBRACE;
               break;
             case ']':
               currentToken = RBRACE;
               break;
             case ',':
               currentToken = COMMA;
               break;
             default:
               currentToken = ERROR;
               break;
           }
         }
         break;
```

state가 start 에서 시작할때, 위에서 선언한 enum에 맞춰서 currentToken을 지정해 준다.

```
/ for ==
       case IFEQ :
         state = DONE;
         if (c == '=') currentToken = EQ;
         else
         {
           ungetNextChar();
           currentToken = ASSIGN;
```

```
        }
        break;
      // for =<
       case IFLT :
        state = DONE;
        if (c == '=') currentToken = LE;
        else
        {
          ungetNextChar();
          currentToken = LT;
        }
        break;
      // for >=
      case IFGT :
        state = DONE;
        if (c == '=') currentToken = GE;
        else
        { ungetNextChar();
          currentToken = GT;
        }
        break;
      // for !=
      case IFNE:
        state = DONE;
        if (c == '=') currentToken = NE;
        else
        { ungetNextChar();
          save = FALSE;
          currentToken = ERROR;
        }
        break;
      // for /* */ INMIDCOMMENT ~ INCOMMENT
       case COMMENTTEXT :
        if (c == '*')
         {
           state = INCOMMENT;
           save = FALSE;
         }
         else
         {
           state = DONE;
           ungetNextChar();
           currentToken = OVER;
         }
         break;
       case INCOMMENT:
         save = FALSE;
         if (c == EOF)
         { state = DONE;
           currentToken = ENDFILE;
         }
         else if (c == '*') state = MIDCOMMENT;
         break;
       case MIDCOMMENT:
         save = FALSE;
         if (c == EOF)
         {
           state = DONE;
           currentToken = ENDFILE;
         }
         else if (c == '/') state = START;
         else if (c == '*') state = MIDCOMMENT;
         else state = INCOMMENT;
         break;
       case INNUM:
         if (!isdigit(c))
         { /* backup in the input */
           ungetNextChar();
           save = FALSE;
           state = DONE;
           currentToken = NUM;
         }
         break;
       case INID:
         if (!isalpha(c))
         { /* backup in the input */
           ungetNextChar();
           save = FALSE;
           state = DONE;
           currentToken = ID;
         }
         break;
       case DONE:
       default: /* should never happen */
         fprintf(listing,"Scanner Bug: state= %d\n",state);
         state = DONE;
         currentToken = ERROR;
```

```
            break;
        }
```

state가 변함에 따라 case문에서 해당 state에 맞는 역할을 진행한다. ==, > =, < =, ! =, /* */ 의 경우는 token을 하나만 받아서는 알 수 없으므로 case 문안에 if문을 만들어 그 다음 token이 무엇인지에 따라서 다르게 진행한다.

주석의 경우 INCOMMENT 로 주석을 들어가고 IFCOMMENT가 있다면 주석임을 판단하여 MIDCOMMENT 로 주석이 끝났음을 판단한다.

```
make cminus_cimpl
./cminus_cimpl test1.cm
```

위의 두 명령어를 terminal 에 각각 입력하여 실행 결과를 확인한다.

test1.cm 의 결과

```
TINY COMPILATION: test1.cm
  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 1 : CMINUS_LEX

```
"if"            {return IF;}
"else"          {return ELSE;}
"="             {return ASSIGN;}
"=="            {return EQ;}
"<"             {return LT;}
"+"             {return PLUS;}
"-"             {return MINUS;}
"*"             {return TIMES;}
"/"             {return OVER;}
"("             {return LPAREN;}
")"             {return RPAREN;}
";"             {return SEMI;}

//추가한 keyword와 symbol
"int"           {return INT;}
"void"          {return VOID;}
"while"         {return WHILE;}
"return"        {return RETURN;}
">"             {return GT;}
"<="            {return LE;}
">="            {return GE;}
"!="            {return NE;}
"{"             {return LCURLY;}
"}"             {return RCURLY;}
"["             {return LBRACE;}
"]"             {return RBRACE;}
","             {return COMMA;}

"/*"             { char c;
                   char prevC = '\0';
                   do
                   { c = input();
                     if (c == EOF) break;
                     if (c == '\n') lineno++;
                     if (prevC == '*' && c == '/') break;
                     prevC = c;
                   } while (1);
                 }
{number}        {return NUM;}
{identifier}    {return ID;}
{newline}       {lineno++;}
{whitespace}    {/* skip whitespace */}
.               {return ERROR;}
```

flex를 사용하여 C minus의 lexer를 자동으로 생성해주기 때문에 Keyword와 Symbol만 추가한다.

int, void, while, return, >, < =, > =, ! =, {, }, [,], , ,/* 를 추가해준다.

이때 주석의 경우 두 문자열을 비교해야 하므로, 현재 문자열을 저장하는 char c 와 이전의 문자열을 저장하는 char pervC 를 만들어 주석을 처리한다.

```
make cminus_lex
./cminus_lex test1.cm
```

위 두 명령어를 입력하여 위의 결과와 동일한지 확인한다.

```
make cminus_cimpl cminus_lex
```

위의 명령어와 같이 두 파일을 동시에 complie하여 결과를 확인 할 수 있다.