

Compiler Project01_Scanner Report

2015004693_양상현

Goal:

- Implementation of C-MINUS Scanner

Process:

- (1). By editing source codes "globals.h", "main.c", "util.h", "util.c", "scan.h", "scan.c"
- (2). By using flex by editing "tiny.l" to "cminus.l"
- (3). Creating both "cminus_cimpl" , "cminus_flex" execution files by editing "Makefile"

(1).

일단 과제 명세 pdf에 나온 hint 대로 "globals.h" 에서 "MAXRESERVED"를 8에서 12로 바꿔주고, "TokenType"에 새로운 enum 변수들을 추가해 주었고, "main.c"에서 NO_PARSE, EchoSource, TraceScan을 TRUE로 바꿔주었고, "scan.c"에서 'StateType'과 'reservedWords[]'를 편집해주었다. 또한 "util.c" 에서 printToken()함수도 새로 추가된 tokentype들을 다 추가하여 편집해주었다.

가장 시간을 많이 할애하여 편집한 부분은 "scan.c"에서 'getToken()'함수의 내부 이다. 새로 추가된 StateType 에서의 경우의 수를 다 생각하여 DFA를 편집해주었다.

'=', '<', '>', '/', '!' 를 읽을 경우 각각 ASSIGN or EQ, LT or LE, GT or GE, OVER or INCOMMENT, NE or ERROR 가 될 수 있기에 이러한 경우를 잘 고려하여 DFA를 설계하였다. 또한 INCOMMENT상태에서 '*'를 읽게되면 INCOMMENT_ 상태가 되고 이때 '/'를 읽게되면 START, 다른 걸 일게 되면 다시 INCOMMENT 상태로 가게 된다.

편집된 < getToken() in scan.c>, 빨간색 글자가 변경된 부분:

<pre>(1)./* function getToken returns the * next token in source file*/ 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 == ' ') (c == '\t') (c == '\n')) save = FALSE; else if(c == '='){ save = FALSE; state = INEQ; } else if(c == '<'){ save = FALSE; state = INLT; </pre>	<pre>(3) break; case INEQ: //have read '=' state = DONE; if(c == '='){ currentToken = EQ; } else{ ungetNextChar(); save = FALSE; currentToken = ASSIGN; } break; case INLT: //have read '<' state = DONE; if(c == '='){ currentToken = LE; } else{ ungetNextChar(); save = FALSE; currentToken = LT; } break; case INGT: //have read '>' state = DONE; if(c=='='){ currentToken = GE; } </pre>
--	--

<pre> (2) } else if(c == '>'){ save = FALSE; state = INGT; } else if(c == '!'){ save = FALSE; state = INNE; } else if(c == '/'){ 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; 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; } </pre>	<pre> (4) else{ ungetNextChar(); save = FALSE; currentToken = GT; } break; case INNE: //have read '!' state = DONE; if(c == '='){ currentToken = NE; } else{ ungetNextChar(); save = FALSE; currentToken = ERROR; } break; case INOVER: // have read '/' if(c == '*'){ state = INCOMMENT; save = FALSE; } else { state = DONE; ungetNextChar(); save = FALSE; currentToken = OVER; } break; case INCOMMENT: save = FALSE; if (c == EOF) { state = DONE; currentToken = ENDFILE; } else if (c == '*'){ state = INCOMMENT_; } break; case INCOMMENT_: // have read '/*---*' if(c == '/'){ state = START; save = FALSE; } else if(c == '*'){ state = INCOMMENT_; save = FALSE; } else if(c == EOF){ state = DONE; save = FALSE; currentToken = ENDFILE; } else{ state = INCOMMENT; save = FALSE; } break; case DONE: default: /* should never happen */ fprintf(listing,"Scanner Bug: state= %d\n",state); state = DONE; currentToken = ERROR; break; } 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; } /* end getToken */ </pre>
--	---

(2).

"tiny.l"을 "cminus.l"로 편집하여 "flex"를 사용하여 "scan.c"의 역할을 하는 "lex.yy.c"를 생성한다. 이 부분에서 중요한 것은 아래 코드에서 빨간색으로 표시된 부분, NFA의 state를 나타내어 주는 부분을 잘 편집하는 것인데, 'COMMENT'("/...*/") state를 제외한 나머지 부분은 단순히 TokenType를 return 하는 방식으로 간단히 구현이 되었다.

COMMENT(/...*/)를 구현하는 부분에서 Nested comments(/* /* /* /* /* /* /*)를 잘 인식할 수 있게 하는 부분을 구현할 때 시간이 매우 많이 걸렸다. (이 부분은 밑에 cminus.l 에서 파란색 글씨로 되어있다)

"/*"를 읽고 INCOMMENT state로 들어왔을때, *를 읽으면 INCOMMENT_state 가 되고 여기서 '/'을 읽으면 START state가 되고, '/'이 아닌 다른것('\\\\n' '\\\\t' ' ' EOF 포함)을 읽게 되면 다시 INCOMMENT state 가 되게끔 만들어주었다. 이렇게 해줌으로써 '/' 다음에 계속해서 '/'이 나오더라도 이를 단순 주석으로 인식하고, 가장 먼저 나오는 '*'로 INCOMMENT state를 벗어나게 된다. Ex("/ * a / * b / * c / d * / e * /" 에서 **"/ * a / * b / * c / d * / e * /"** 파란색 부분만 주석으로 인식이 되고 빨간색 부분은 일반 token Stream으로 인식이 된다.)

편집된<cminus.l>, 빨간색 파란색 글자가 변경된 부분:

<pre>(1) /*****/ /* File: tiny.l */ /* Lex specification for TINY */ /* Compiler Construction: Principles and Practice */ /* Kenneth C. Louden */ /*****/ %{ #include "globals.h" #include "util.h" #include "scan.h" /* lexeme of identifier or reserved word */ char tokenString[MAXTOKENLEN+1]; %} digit [0-9] number {digit}+ letter [a-zA-Z] identifier {letter}+ newline \\n whitespace [\\t]+ %% "if" {return IF;} "then" {return THEN;} "else" {return ELSE;} "while" {return WHILE;} "return" {return RETURN;} "int" {return INT;} "void" {return VOID;} "end" {return END;} "repeat" {return REPEAT;} "until" {return UNTIL;} "read" {return READ;} "write" {return WRITE;} "=" {return ASSIGN;} "==" {return EQ;} "!=" {return NE;} "<=" {return LE;} ">=" {return GE;} ">" {return GT;} "<" {return LT;} "+" {return PLUS;} "_" {return MINUS;} "*" {return TIMES;} "/" {return OVER;} "(" {return LPAREN;} ")" {return RPAREN;} "{" {return LBRACE;}</pre>	<pre>(2) "}" {return RBRACE;} "{" {return LCURLY;} "}" {return RCURLY;} ";" {return SEMI;} "," {return COMMA;} {number} {return NUM;} {identifier} {return ID;} {newline} {lineno++;} {whitespace} {/* skip whitespace */} "/*" { char c = ' '; while(1){ while(c != '*'){ c = input(); if(c == EOF) break; if(c == '\\n') lineno++; } //shoud be c == "*" c = input(); if(c == EOF) break; if(c == '\\n') lineno++; if(c == '/') break; } } . {return ERROR;} %% TokenType getToken(void) { static int firstTime = TRUE; TokenType currentToken; if (firstTime) { firstTime = FALSE; lineno++; yyin = source; yyout = listing; } currentToken = yylex(); strncpy(tokenString,yytext,MAXTOKENLEN); if (TraceScan) { fprintf(listing,"\\t%d: ",lineno); printToken(currentToken,tokenString); } return currentToken; }</pre>
--	---

(3).

Makefile 에서 "make all" command를 이용하여 "cminus_cimpl", "cminus_flex" 실행파일들을 동시에 생성할수 있게끔 하기 위해 Makefile을 다음과 같이 편집해주었다.

편집된 <Makefile>, 빨간색 글자가 변경된 부분:

```
CC = gcc
CFLAGS =
OBSJ = main.o util.o scan.o parse.o symtab.o analyze.o code.o cgen.o
OBSJ_FLEX = main.o util.o lex.yy.o
cminus_flex: $(OBSJ_FLEX)
    $(CC) $(CFLAGS) $(OBSJ_FLEX) -o cminus_flex -lfl
cminus_cimpl: $(OBSJ)
    $(CC) $(CFLAGS) $(OBSJ) -o cminus_cimpl
main.o: main.c globals.h util.h scan.h parse.h analyze.h cgen.h
    $(CC) $(CFLAGS) -c main.c
util.o: util.c util.h globals.h
    $(CC) $(CFLAGS) -c util.c
lex.yy.o: cminus.l scan.h util.h globals.h
    flex cminus.l
    $(CC) $(CFLAGS) -c lex.yy.c -lfl
scan.o: scan.c scan.h util.h globals.h
    $(CC) $(CFLAGS) -c scan.c
parse.o: parse.c parse.h scan.h globals.h util.h
    $(CC) $(CFLAGS) -c parse.c
symtab.o: symtab.c symtab.h
    $(CC) $(CFLAGS) -c symtab.c
analyze.o: analyze.c globals.h symtab.h analyze.h
    $(CC) $(CFLAGS) -c analyze.c
code.o: code.c code.h globals.h
    $(CC) $(CFLAGS) -c code.c
cgen.o: cgen.c globals.h symtab.h code.h cgen.h
    $(CC) $(CFLAGS) -c cgen.c
clean:
    -rm cminus_cimpl      #for scan.c
    -rm tm                #for both
    -rm cminus_flex      #for flex
    -rm $(OBSJ) #for both
tm: tm.c
    $(CC) $(CFLAGS) tm.c -o tm
all: cminus_cimpl tm cminus_flex
```

Results:

By "cminus_cimpl" scanning "test.cm" "./cminus_cimpl test.cm"	By "cminus_flex" scanning "test.cm" "./cminus_flex test.cm"
<pre> C-MINUS COMPILATION: test.cm 1: /*A program to perform Euclid's 2: Algorithm to compute 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= v 7: ID, name= v 7:) 7: ; 8: ; 9: /* u-u/v*v == u mod v */ 9: } 10: </pre>	<pre> C-MINUS COMPILATION: test.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= v 7: * 7: ID, name= v 7:) 9: } </pre>
<p>과제 명세 pdf 내에 있는 예시 코드 test.cm을 scan 한 결과 위와 같은 출력 화면을 볼 수 있었다. 위 출력된 화면을 과제 명세 pdf 내에 있는 scan 결과 슬라이드와 비교해본 결과 모두 일치하였다.</p>	<p>- 또한 test.cm 의 주석 부분을 중첩으로 사용하여 실행해본 결과, 구현목표대로 <code>"/* a /* b /* c /* d /* e */"</code> 에서 <code>"/* a /* b /* c /* d /* e */"</code> 파란색 부분만 주석으로 인식이 되고 빨간색 부분은 일반 token Stream으로 인식이 되는 결과를 확인할 수 있었다.</p>