Scanner

컴퓨터전공

2013011822

김병조

Modification Tiny compiler code (globals.h, main.c, util.c)

globals.h

```
#define MAXRESERVED 12

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

C-Minus의 새로운 reserved word (ELSE, WHILE, RETURN, INT, VOID)와 special symbol (NE, LE, GT, GE, LBRACE, RBRACE, LCURLY, RCURLY, COMMA) 를 TokenType에 추가한다. 그리고 그에 따라 MAXRESERVED 값을 수정한다. Tiny 컴파일러의 reserved words 는 아직 삭제하지 않은 상태이다. 이는 다음 과제 때 삭제 하려고 한다.

main.c

```
#define NO_PARSE TRUE
```

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

scanner-only compiler 와 tracing flags를 위해 flag 들을 수정한다.

util.c

```
void printToken( TokenType token, const char* tokenString )
{ switch (token)
  { case IF:
    case ELSE:
    case WHILE:
    case RETURN:
    case INT:
    case VOID:
    case THEN:
    case END:
    case REPEAT:
    case UNTIL:
    case READ:
    case WRITE:
      fprintf(listing,
         "reserved word: %s\n",tokenString);
      break;
    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;
```

printToken에 case를 추가한다. ELSE, WHILE, RETURN, INT, VOID 라는 토큰이 들어올 경우 "reserved word: "를 출력 하도록 하고 NE, LE, GT, GE, LBRACE, RBRACE, LCURLY, RCURLY인 경우 각각의 문자열을 출력 하도록 한다.

Implementation of C-Scanner using C-code by scan.c modification

scan.c

```
typedef enum
   { START,INEQ,INCOMMENT,INNUM,INID,DONE,INLT,INGT,INNE,INOVER,INCOMMENT_ }
   StateType;
```

DFA 사용하기 위한 state를 추가한다. <=, < >=, >, !=, ==, =, /, /**/ 를 사용하기 위해 스테이트를 추가한 모습이다.

```
static struct
    { char* str;
      TokenType tok;
    } reservedWords[MAXRESERVED]
   = {{"if",IF},{"else",ELSE},{"while",WHILE},{"return",RETURN},{"int",INT},{"void",VOID},
      {"then",THEN},{"end",END},
      {"repeat",REPEAT},{"until",UNTIL},{"read",READ},
      {"write",WRITE}};
```

추가된 reservedwords를 위해 스트링 값과 토큰타입을 매치 시켜 놓는다.

getToken() 함수는 문자열을 하나 받고 그것의 토큰을 판단해서 return 하는 함수 이다. START state에서 시작해서 우선 숫자, 알파벳인지, 공백, 탭, ₩n 를 판단 한다. /**/와 /, ==와 =, <와 <=, >와 >= 그리고 !=를 판단 하기 위해 문자열이 들어 왔을 때 그거에 맞는 state로 변환을 시킨다. 그 후 다음으로 들어오는 문자열을 이용하여 토큰을 판단한다. 나머지 symbols들은 switch case 문을 이용하여 토큰을 리턴 한다.

// or / 판단**

```
else if (c == '/')
{
  state = INOVER;
}
```

state가 START 일 때 문자열에 / 가 들어오면 state를 INOVER로 바꾸고 다음 문자열을 받는다.

```
case INOVER:
  if( c == '*' ) state = INCOMMENT;

  else
  {
   ungetNextChar();
   save = FALSE;
   currentToken = OVER;
   state = DONE;
  }
  break;
```

INOVER state에서 문자열 * 이 들어오면 /* 인 것으로 확인하고 state를 INCOMMENT로 바꾼다. 하지만 다른 문자열이 들어오면 / 를 over로 판단을 하여 currentToken을 OVER로 설정한다. 이때 문자열을 한번 더 받았기 때문에 ungetNextChar()를 호출 하고 state를 DONE으로 설정하여 다음 문자열을 다시 받도록 한다.

```
case INCOMMENT:
  save = FALSE;
  if (c == EOF)
  { state = DONE;
    currentToken = ENDFILE;
  }
  else if ( c == '*' ) state = INCOMMENT_;
  break;

case INCOMMENT_:
  save = FALSE;
  if (c == EOF)
  { state = DONE;
    currentToken = ENDFILE;
  }
  else if (c == '/') state = START;
  else state = INCOMMENT;
  break;
```

INCOMMENT state에서는 * 이 들어올 때 까지 문자열을 받는다. * 이 들어오면 state를 INCOMMENT_로 바꾼다. INCOMMENT_에서는 곧이어 / 라는 문자열이 들어오는지 확인한 후 /**/ 주석의 완성을 판단 한 후 state를 START로 바꾼다. / 문자열이 바로 안 들어올 경우 다시 state 를 INCOMMENT로 바꾸어 * 를 기다린다.

= or == 판단

```
else if (c == '=')
{
 state = INEQ;
}
```

state가 START 일 때 문자열에 = 가 들어오면 state를 INEQ로 바꾸고 다음 문자열을 받는다.

```
case INEQ:
  if( c == '=' ) currentToken = EQ;
  else{
   ungetNextChar();
   save = FALSE;
   currentToken = ASSIGN;
  }
  state = DONE;
  break;
```

INEQ state에 =가 들어오면 == 인 것으로 판단 한 후 currentToken 을 EQ로 설정 한다. 다른 문자열이 들어오면 ungetNextChar()를 호출 후 = 인 것으로 판단 해서 currentToken을 ASSIGN으로 설정 한다. 토큰이 결정 되어 졌기 때문에 state는 DONE으로 설정한다.

< or <=, > or >= 판단

```
else if (c == '<')
{
 state = INLT;
}
```

state가 START 일 때 문자열에 < 가 들어오면 state를 INLT로 바꾸고 다음 문자열을 받는다.

```
case INLT:
  if (c == '=') currentToken = LE;
  else{
   ungetNextChar();
   save = FALSE;
   currentToken = LT;
  }
  state = DONE;
  break;
```

INLT state에 = 가 들어오면 <= 인 것으로 판단 한 후 currentToken 을 LE로 설정 한다. 다른 문자열이 들어오면 ungetNextChar()를 호출 후 < 인 것으로 판단 해서 currentToken을 LT로 설정 한다. >(GT) 와 >=(GE) 또한 이런 방법으로 판단한다. 토큰이 결정 되어 졌기 때문에 state는 DONE으로 설정한다.

!= 판단

```
else if (c == '!')
{
 state = INNE;
}
```

state가 START 일 때 문자열에 ! 가 들어오면 state를 INNE로 바꾸고 다음 문자열을 받는다.

```
case INNE:
  if (c == '=') currentToken = NE;
  else{
   ungetNextChar();
   save = FALSE;
   currentToken = ERROR;
  }
  state = DONE;
  break;
```

INNE state에 = 가 들어오면 != 인 것으로 판단 한 후 currentToken 을 NE로 설정 한다. 다른 문자열이 들어오면 ungetNextChar()를 호출 후 error 인 것으로 판단 해서 currentToken을 ERROR로 설정 한다. 토큰이 결정 되어 졌기 때문에 state는 DONE으로 설정한다.

나머지 토큰 판단

```
case '{':
  currentToken = LCURLY;
  break;
case '}':
  currentToken = RCURLY;
  break;
case '[':
  currentToken = LBRACE;
  break;
case ']':
  currentToken = RBRACE;
```

나머지 토큰은 state가 START 일 때 switch case 문을 이용하여 들어오는 문자열에 따라 토큰 설정을 한다.

Makefile – Makefile 첨부

주어진 Makefile을 수정 없이 실행 시켜도 된다. make tiny 명령어를 이용하여 tiny 파일을 만든다.

Implementation of C-Scanner using lex(flex) by Tiny.l modification

cimus.l

getToken()함수에 yylex() 함수를 이용하여 rule section 의 문자열을 비교하여 토큰을 리턴하여 currentToken을 설정한다.

```
"="             {return ASSIGN;}
"=="            {return EQ;}
"!="            {return NE;}
"<"             {return LT;}
"<="            {return LE;}
">"             {return GT;}
">="            {return GE;}
"("             {return LPAREN;}
")"             {return RPAREN;}
"["             {return LBRACE;}
"]"             {return RBRACE;}
"{"             {return LCURLY;}
"}"             {return RCURLY;}
```

rule section 에 새로 세팅 할 스트링과 그에 따른 토큰타입을 설정한다.

// 판단**

```
"/*"            { char c;
                  char prev = '\0';
                  do
                  { c = input();
                    if (c == EOF) break;
                    if (c == '\n') lineno++;
                    if (prev == '*' && c == '/') break;
                    prev = c;
                  } while (1);
                }
```

문자열에 /* 가 들어올 경우 */ 이 들어오기를 기다린다. */ 는 두개의 문자가 연속되어 있기 때문에 prev를 이용하여 (prev == "*" && c == '/') 가 만족 할 때 /* */ 의 완성을 판단한다.

Makefile – Makefile 첨부

```
cminus_flex: $(OBJS_FLEX)
    $(CC) $(CFLAGS) main.o util.o lex.yy.o -o cminus_flex -lfl

lex.yy.o: cminus.l scan.h util.h globals.h
    flex cminus.l
    $(CC) $(CFLAGS) -c lex.yy.c -lfl
```

flex 를 이용하여 cminus.l 로 lex.yy.c 을 만든다. 그 후 gcc 명령어를 이용하여 cminus_flex 파일을 만든다. 이때 -lfl 옵션을 추가하여 cminus.l 이 컴파일 되도록 한다. 그 후 $make cminus_flex 명령어를 이용하여 cminuns_flex 파일을 만든다. 이때 Makefile 상단에 OBJS_FLEX = main.o util.o lex.yy.o 를 추가한다.

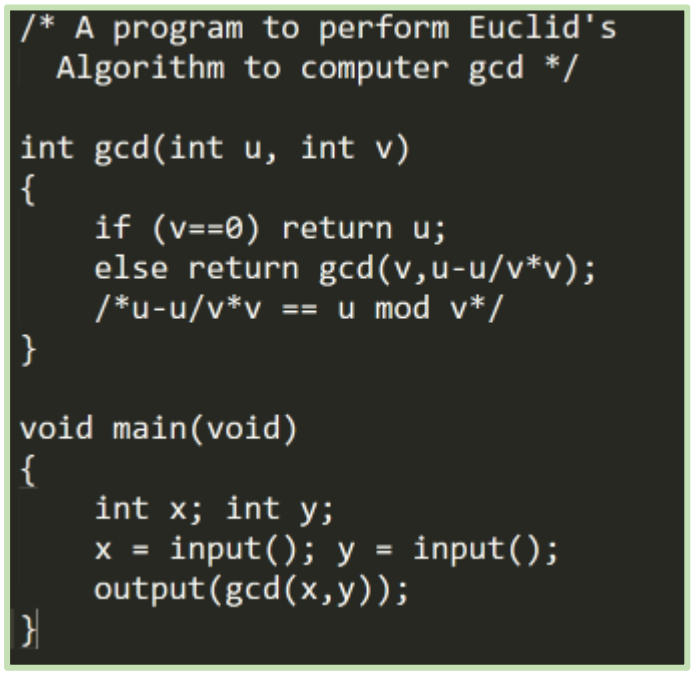
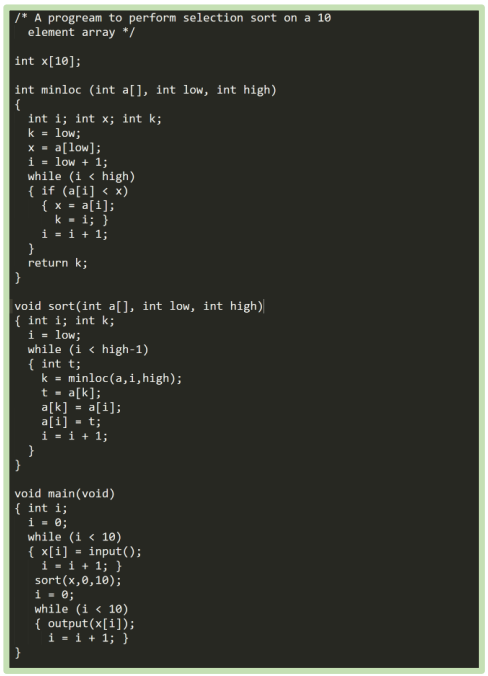
Compilation environment

Os : Ubuntu 16.04.3 LTS

gcc : gcc (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609

Example and Result

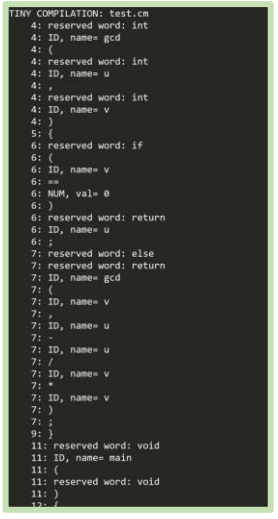
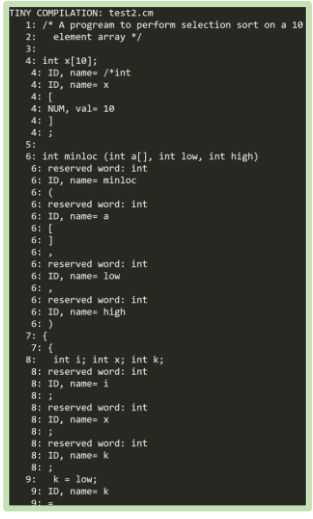
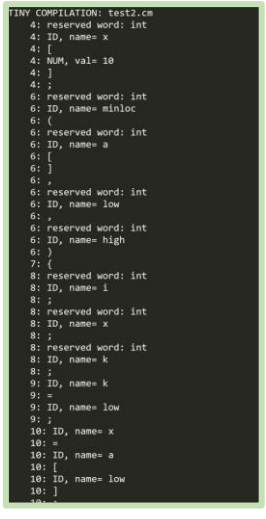
Example : ./result/gcd.cm, ./result/sort.cm 첨부

gcd.cm	sort.cm
	

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

```
/* A program to perform selection sort on a 10
  element array */

int x[10];

int minloc (int a[], int low, int high)
{
  int i; int x; int k;
  k = low;
  x = a[low];
  i = low + 1;
  while (i < high)
  { if (a[i] < x)
    { x = a[i];
      k = i; }
    i = i + 1;
  }
  return k;
}

void sort(int a[], int low, int high)
{ int i; int k;
  i = low;
  while (i < high-1)
  { int t;
    k = minloc(a,i,high);
    t = a[k];
    a[k] = a[i];
    a[i] = t;
    i = i + 1;
  }
}

void main(void)
{ int i;
  i = 0;
  while (i < 10)
  { x[i] = input();
    i = i + 1; }
  sort(x,0,10);
  i = 0;
  while (i < 10)
  { output(x[i]);
    i = i + 1; }
}
```

Result : ./result/gcd_with_tiny, ./result/gcd_with_cminus, ./result/sort_with_tiny, ./result/sort_with_cminus 첨부

gcd_with_tiny	gcd_with_cminus	sort_with_tiny	sort_with_cminus
			

```
TINY COMPILATION: test.cm
  1: /* A program to perform Euclid's
  2:    Algorithm to computer gcd */
  3:
  4: int gcd(int u, int v)
   4: ID, name= /*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= u
   7: /
   7: ID, name= v
   7: *
   7: ID, name= v
   7: )
   7: ;
```

```
TINY 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= 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: )
```

```
TINY COMPILATION: test2.cm
  1: /* A program to perform selection sort on a 10
  2:   element array */
  3:
  4: int x[10];
   4: ID, name= /*int
   4: ID, name= x
   4: [
   4: NUM, val= 10
   4: ]
   4: ;
  5:
  6: int minloc (int a[], int low, int high)
   6: reserved word: int
   6: ID, name= minloc
   6: (
   6: reserved word: int
   6: ID, name= a
   6: [
   6: ]
   6: ,
   6: reserved word: int
   6: ID, name= low
   6: ,
   6: reserved word: int
   6: ID, name= high
   6: )
   7: {
  8:   int i; int x; int k;
   8: reserved word: int
   8: ID, name= i
   8: ;
   8: reserved word: int
   8: ID, name= x
   8: ;
   8: reserved word: int
   8: ID, name= k
   8: ;
   9:   k = low;
   9: ID, name= k
```

```
TINY COMPILATION: test2.cm
   4: reserved word: int
   4: ID, name= x
   4: [
   4: NUM, val= 10
   4: ]
   4: ;
   6: reserved word: int
   6: ID, name= minloc
   6: (
   6: reserved word: int
   6: ID, name= a
   6: [
   6: ]
   6: ,
   6: reserved word: int
   6: ID, name= low
   6: ,
   6: reserved word: int
   6: ID, name= high
   6: )
   7: {
   8: reserved word: int
   8: ID, name= i
   8: ;
   8: reserved word: int
   8: ID, name= x
   8: ;
   8: reserved word: int
   8: ID, name= k
   8: ;
   9: ID, name= k
   9: =
   9: ID, name= low
   9: ;
  10: ID, name= x
  10: =
  10: ID, name= a
  10: [
  10: ID, name= low
  10: ]
```