Project1 Scanner

2019094511 김준표

1. Compilation method and environment

- Environment: Windows 10 WSL2 – Ubuntu 18.04

2. Explanation about how to implement and how to operate

- Method 1: "cminus_cimpl"

첫 번째 방법으로는 DFA를 이용하여 token을 인식하는 방법이다. globals.h에서 token을 정의하고 scan.c에서 이를 이용하여 scanner를 생성한다. 구현하는 scanner는 input으로 받은 코드를 tokenize한다.

<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;
```

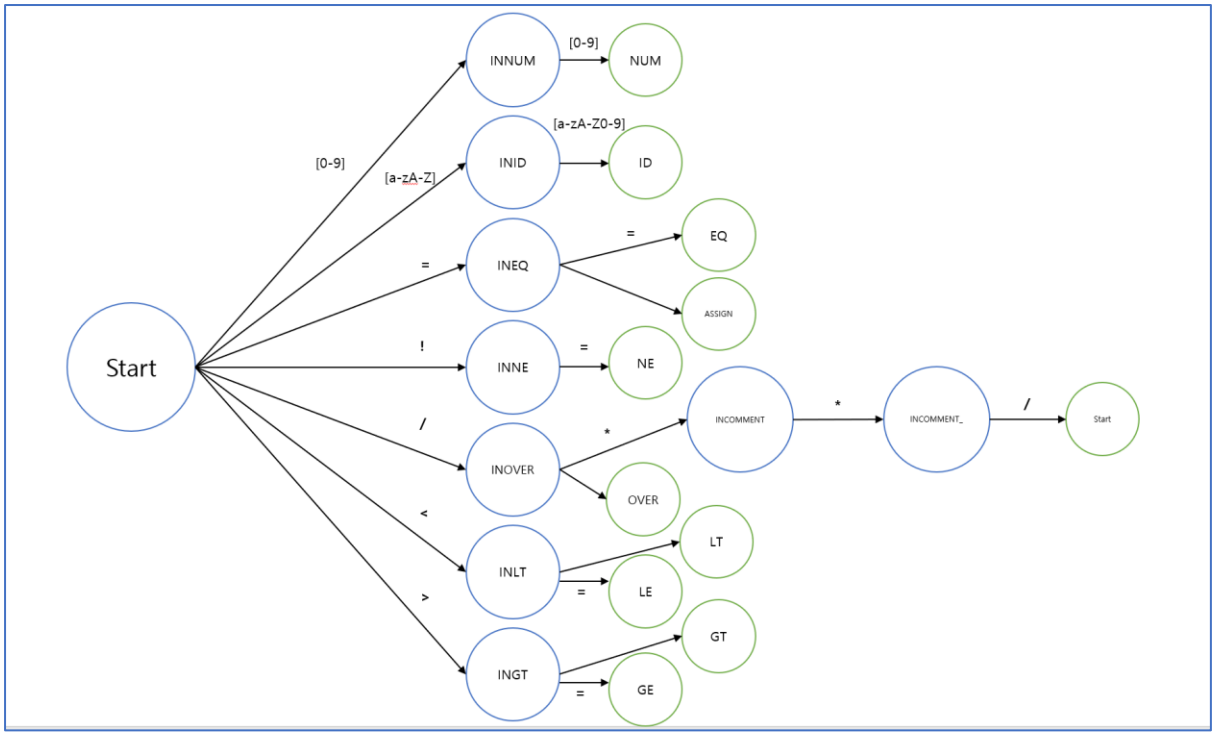
globals.h에서는 이와 같이 token을 정의한다.

<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;
    //TODO
    //ASSIGN,EQ,NE,LT,LE,GT,GE,PLUS,MINUS,TIMES,OVER,
    //LPAREN,RPAREN,LBRACE,RBRACE,LCURLY,RCURLY,SEMI,COMMA
```

util.c의 printToken함수는 tokenize된 결과를 출력한다. 해당 함수에서는 keyword와 symbol을 출력하는 양식에 맞게 출력하도록 한다.

<scan.c>



이와 같이 표현되는 DFA를 따라 tokenize를 실시한다. 특별히 주의할 점은 다음과 같다.

1. INID에서 (letter|digit)*을 받으면 ID로 넘어가는 부분 체크
2. Multi character symbol의 처리 방안
3. /* */와 같은 주석의 처리를 위한 INCOMMENT, INCOMMENT_ state의 흐름 체크

위의 세 가지 부분을 주의하며 DFA state의 흐름을 이어주면 된다.

```
case INID:
  if (!(isalpha(c)||isdigit(c)))
  { /* backup in the input */
    ungetNextChar();
    save = FALSE;
    state = DONE;
    currentToken = ID;
  }
  break;
case INEQ:
  state = DONE;
  if (c == '=')
    currentToken = EQ;
  else
  { /* backup in the input */
    ungetNextChar();
    save = FALSE;
    currentToken = ASSIGN;
  }
  break;
```

```
case INOVER:
  // Case: /* */
  if (c == '*')
  {
    save = FALSE;
    state = INCOMMENT;
  }
  else
  {
    state = DONE;
    ungetNextChar();
    currentToken = OVER;
  }
  break;
case INCOMMENT:
  save = FALSE;
  if (c == EOF)
  { state = DONE;
    currentToken = ENDFILE;
  }
  // Case: /* abc * -> goto INCOMMENT_ state
  else if (c == '*') state = INCOMMENT_;
  break;
case INCOMMENT_:
  save = FALSE;
  if (c == EOF)
  { state = DONE;
    currentToken = ENDFILE;
  }
  // Case: /* abc */ -> End comment and go to START state
  else if (c == '/') state = START;
  // Case: /* abc ** -> Wait for '/' in INCOMMENT_ state
  else if (c == '*') state = INCOMMENT_;
  // Case: /* abc *a -> Still comment so go to INCOMMENT state
  else state = INCOMMENT;
  break;
```

위의 코드는 multi character와 주석 처리를 위한 INOVER → INCOMMENT → INCOMMENT_ state의 흐름을 제어한 코드의 일부분이다. 이와 같이 scan.c에서는 input으로 받는 코드를 구문에 맞게 tokenize하여 반환해주는 역할을 한다.

- Method 2: “cminus_lex”

두 번째 방법으로는 Regular expression을 이용하는 방법이다. 이는 Lex 코드를 작성하여 lexical pattern을 구현한다. 이는 cminus.l에서 구현했다.

```
digit       [0-9]
number      {digit}+
letter      [a-zA-Z]
identifier  [a-zA-Z][a-zA-Z0-9]*
newline     \n
whitespace  [ \t]+
```

해당 부분에서는 ID를 표현하기 위한 identifier 부분을 {letter}+ 에서 letter 이후에 digit이 나올 수 있도록 수정해줬다. regular expression을 활용하여 [a-zA-Z][a-zA-Z0-9]*로 letter(letter|digit)*을 accept 할 수 있도록 했다.

```
"/*"            { char c;
                  int flag = 0;
                  while(1)
                  { c = input();
                    if (c == EOF) break;
                    if (c == '\n') lineno++;
                    if (flag == 1 && c == '/') break;
                    if (c == '*') flag = 1;
                    else flag = 0;
                  };
                }
.               {return ERROR;}
```

또한 multiline 주석 처리를 위해 flag를 이용하여 /* */를 인식할 수 있도록 조건을 추가했다.

3. Example and Result Screenshot

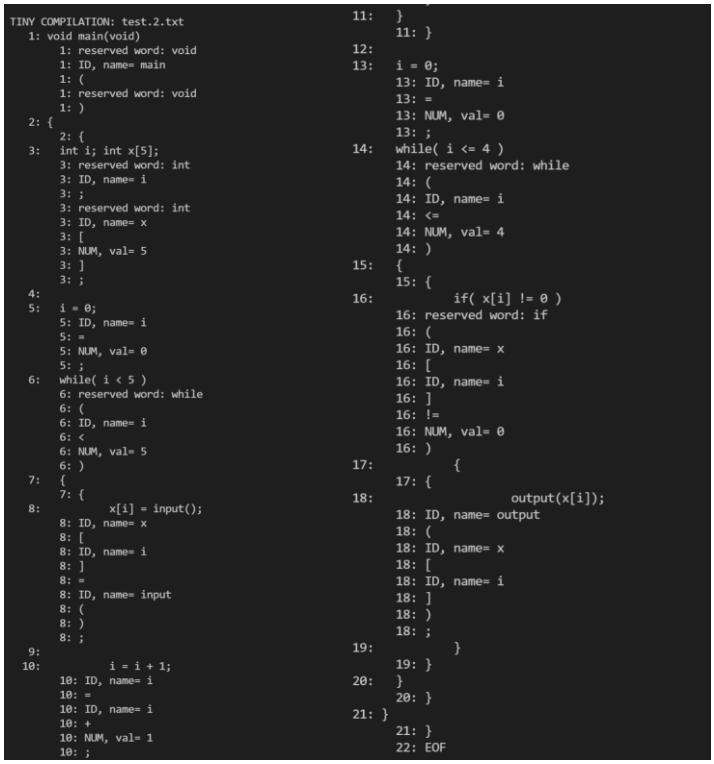
1. cminus_cimpl 테스트

test.1.txt

```
TINY COMPILATION: test.1.txt
   1: /* A program to perform Euclid's
   2:    Algorithm to computer 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= u
        7: /
        7: ID, name= v
        7: *
        7: ID, name= v
        7: )
        7: ;
   8:   /* u-u/v*v == u mod v */
   9: }
        9: }
  10:
```

```
  11: void main(void)
        11: reserved word: void
        11: ID, name= main
        11: (
        11: reserved word: void
        11: )
  12: {
        12: {
  13:   int x; int y;
        13: reserved word: int
        13: ID, name= x
        13: ;
        13: reserved word: int
        13: ID, name= y
        13: ;
  14:   x = input(); y = input();
        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:   output(gcd(x,y));
        15: ID, name= output
        15: (
        15: ID, name= gcd
        15: (
        15: ID, name= x
        15: ,
        15: ID, name= y
        15: )
        15: )
        15: ;
  16: }
        16: }
        17: EOF
```

test.2.txt



```
TINY COMPILATION: test.2.txt
  1: void main(void)
       1: reserved word: void
       1: ID, name= main
       1: (
       1: reserved word: void
       1: )
  2: {
       2: {
  3:   int i; int x[5];
       3: reserved word: int
       3: ID, name= i
       3: ;
       3: reserved word: int
       3: ID, name= x
       3: [
       3: NUM, val= 5
       3: ]
       3: ;
  4:
  5:   i = 0;
       5: ID, name= i
       5: =
       5: NUM, val= 0
       5: ;
  6:   while( i < 5 )
       6: reserved word: while
       6: (
       6: ID, name= i
       6: <
       6: NUM, val= 5
       6: )
  7:   {
       7: {
  8:         x[i] = input();
       8: ID, name= x
       8: [
       8: ID, name= i
       8: ]
       8: =
       8: ID, name= input
       8: (
       8: )
       8: ;
  9:
 10:         i = i + 1;
      10: ID, name= i
      10: =
      10: ID, name= i
      10: +
      10: NUM, val= 1
      10: ;
 11:   }
      11: }
 12:
 13:   i = 0;
      13: ID, name= i
      13: =
      13: NUM, val= 0
      13: ;
 14:   while( i <= 4 )
      14: reserved word: while
      14: (
      14: ID, name= i
      14: <=
      14: NUM, val= 4
      14: )
 15:   {
      15: {
 16:         if( x[i] != 0 )
      16: reserved word: if
      16: (
      16: ID, name= x
      16: [
      16: ID, name= i
      16: ]
      16: !=
      16: NUM, val= 0
      16: )
 17:         {
      17: {
 18:                 output(x[i]);
      18: ID, name= output
      18: (
      18: ID, name= x
      18: [
      18: ID, name= i
      18: ]
      18: )
      18: ;
 19:         }
      19: }
 20:   }
      20: }
 21: }
      21: }
      22: EOF
```

2. cminus_lex 테스트

test.1.txt

```
TINY COMPILATION: test.1.txt
       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
```

test.2.txt

```
TINY COMPILATION: test.2.txt
       1: reserved word: void
       1: ID, name= main
       1: (
       1: reserved word: void
       1: )
       2: {
       3: reserved word: int
       3: ID, name= i
       3: ;
       3: reserved word: int
       3: ID, name= x
       3: [
       3: NUM, val= 5
       3: ]
       3: ;
       5: ID, name= i
       5: =
       5: NUM, val= 0
       5: ;
       6: reserved word: while
       6: (
       6: ID, name= i
       6: <
       6: NUM, val= 5
       6: )
       7: {
       8: ID, name= x
       8: [
       8: ID, name= i
       8: ]
       8: =
       8: ID, name= input
       8: (
       8: )
       8: ;
      10: ID, name= i
      10: =
      10: ID, name= i
      10: +
      10: NUM, val= 1
      10: ;
      11: }
      13: ID, name= i
      13: =
      13: NUM, val= 0
      13: ;
      14: reserved word: while
      14: (
      14: ID, name= i
      14: <=
      14: NUM, val= 4
      14: )
      15: {
      16: reserved word: if
      16: (
      16: ID, name= x
      16: [
      16: ID, name= i
      16: ]
      16: !=
      16: NUM, val= 0
      16: )
      17: {
      18: ID, name= output
      18: (
      18: ID, name= x
      18: [
      18: ID, name= i
      18: ]
      18: )
      18: ;
      19: }
      20: }
      21: }
      22: EOF
```