Implementation of C- Scanner

2017029807 성창호

본 프로그램은 Tiny Compiler와 lex code를 수정하여 C- Scanner를 구동시킨다.

Project Environment

- Ubuntu 16.04.6 (WSL)

Overview

C- Scanner 제작을 위해 C- 문법에 맞게 Keyword, Symbol, Token을 정의하고 DFA를 이용하여 입력받은 문자열을 Token 단위로 Lexical Analysis한다.

Definition

- **keyword (lower case)**
  - if, else, int, return, void, while
- **symbol**
  - + - * / < <= > >= == != = ; , ( ) [ ] { } /* */
- **Token**
  - *ID = letter letter**
  - *NUM = digit digit **
  - *letter* = a | ... | z | A | ... | Z
  - *digit* = 0 | 1 | ... | 9

1. C- Scanner using C code (modify Tiny Compiler)

`main.c`

```
#define NO_PARSE TRUE

int EchoSource = TRUE;
int TraceScan = TRUE;
int TraceParse = FALSE;
int TraceAnalyze = FALSE;
int TraceCode = FALSE;
```

본 프로그램에서는 C- Scanner만 제작하므로 `main.c` 의 flag들을 조정한다.

`globals.h`

```
/* MAXRESERVED = the number of reserved words */
#define MAXRESERVED 12

typedef enum
    /* book-keeping tokens */
   {ENDFILE,ERROR,
    /* reserved words */
    IF,ELSE,WHILE,RETURN,INT,VOID,   /* discarded*/   THEN,END,REPEAT,UNTIL,READ,WRITE,
 s   /* 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- 의 keyword와 symbol들을 `TokenType` enum에 추가해준다. 이 때 `MAXRESERVED` 의 값을 keyword를 추가한 후의 개수인 12개로 변경해준다. symbol의 각 enum 이름은 위의 코드를 참고바란다.

`scan.c`

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

C- Scanner를 위해 DFA를 구성해야 하는데, 기존의 tiny compiler의 state에서 ==, >=, <=, !=, /* */을 위한 state들을 추가해준다.

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

C-의 keyword들도 `reservedwords` 에 추가해준다.

이제 C-의 DFA를 수행하기 위해 위에서 추가해준 state를 기반으로 symbol들을 처리하는 부분을 수정해준다.

```
case START:
         if (isdigit(c))
           state = INNUM;
         else if (isalpha(c))
           state = INID;
         else if (c == '=')
           state = INEQ;
         else if ((c == ' ') || (c == '\t') || (c == '\n'))
           save = FALSE;
         else if (c == '!')
           state = INNE;
         else if (c == '<')
           state = INLT;
         else if (c == '>')
           state = INGT;
         else if (c == '/')
         { save = FALSE;
           state = INOVER;
         }
         else
```

```
          { state = DONE;
            switch (c)
            {
             ...
             /* 한글자로 이루어진 Symbol */
            }
          }
          break;
```

== (EQ), != (NE), <= (LE), >= (GE), /* */ (COMMENT)와 같이 두 글자 이상으로 이루어진 state(symbol)들은 따로 IN~ 형식의 state을 만들어 해당 state로 진입하도록 한다. 그런 경우가 아니라면, 한글자로 이루어진 symbol로 판단할 수 있으므로 해당 symbol에 맞게 처리해준다.

```
case INOVER:
          if (c == '*')
          { state = INCOMMENT;
            save = FALSE;
          }
          else
          { state = DONE;
            ungetNextChar();
            currentToken = OVER;
          }
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;
case INLT:      // '<'
          state = DONE;
          if (c == '=')
            currentToken = LE;
          else
          { ungetNextChar();
            currentToken = LT;
          }
          break;
case INGT:      // '>'
          state = DONE;
          if (c == '=')
            currentToken = GE;
          else
          { ungetNextChar();
            currentToken = GT;
          }
          break;
case INEQ:      // '='
          state = DONE;
          if (c == '=')
            currentToken = EQ;
          else
          { ungetNextChar();
            currentToken = ASSIGN;
          }
          break;
case INNE:      // '!'
          state = DONE;
          if (c == '=')
            currentToken = NE;
          else
          { ungetNextChar();
            save = FALSE;
            currentToken = ERROR;
          }
          break;
```

두 글자 이상으로 이루어진 symbol들을 IN~ state에 속해 있으므로 다음 문자가 해당 symbol에 적합한 문자라면 현재 state을 해당 symbol의 state로 설정한다. 이 때, 주석(comment)는 첫 slash(over)가 나왔을 때와(INOVER->INCOMMENT), 두번째 *(times)가 나왔을 때의 state(INCOMMENT->INCOMMENT_)로 나누어 처리해준다.

How to operate

```
$ make cminus_cimpl
$ ./cminus_cimpl test.cm
```

Result

```
INY COMPILATION: test.cm
   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

TINY COMPILATION: test2.cm
    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
```

2. C- Scanner using lex (flex)

Requirment

```
$ sudo apt-get install flex
```

`cminus.l`

```
"if"            {return IF;}
"else"          {return ELSE;}
"int"           {return INT;}
"return"        {return RETURN;}
"void"          {return VOID;}
"while"         {return WHILE;}
"="             {return ASSIGN;}
"=="            {return EQ;}
"<"             {return LT;}
">"             {return GT;}
"<="            {return LE;}
">="            {return GE;}
"!="            {return NE;}
"+"             {return PLUS;}
"-"             {return MINUS;}
"*"             {return TIMES;}
"/"             {return OVER;}
"("             {return LPAREN;}
")"             {return RPAREN;}
"{"             {return LCURLY;}
"}"             {return RCURLY;}
"["             {return LBRACE;}
"]"             {return RBRACE;}
";"             {return SEMI;}
","             {return COMMA;}
{number}        {return NUM;}
{identifier}    {return ID;}
{newline}       {lineno++;}
{whitespace}    {/* skip whitespace */}
"/*"             { char c;
                  char prev = '\0';
                  do
                  { c = input();
                    if (c == EOF) break;
                    if (c == '\n') lineno++;
                    if (prev == '*' && c == '/') break;
                    prev = c;
                  } while (1);
                }
.               {return ERROR;}
```

flex를 이용하여 C-의 lexer를 자동으로 생성해주기 때문에 C-의 keyword, symbol들만을 추가해주면 된다. 이 때 주석(comment)은 두 글자씩 비교해야 하기 때문에, 이전 문자를 저장하는 `prev` 변수를 선언하여 처리해준다.

Result

```
TINY COMPILATION: test.cm
        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

TINY COMPILATION: test2.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
```