```
 1 // Brandon Chavez, CSCI 450 - Fall 2018, Professor Nelson
 2 /* scan.c:  The scanner for ATL/0. */
 3 // Rated "E", for "Even You!".
 4 /* Get the common definitions and includes. */
 5 #include "defs.h"
 6
 7 /* Extern variables. */
 8 #include "global.h"
 9 //#include "scan.h"
10 #include <stdio.h>
11 #include <ctype.h>
12 /* Storage for the scanner. */
13 // We can use these to compare with strcmp against
   potential matches from src_in.
14 struct reserved
15 {
16     char *name;
17     token value;
18 }  res_words [] = {
19   { "end", END },
20   { "read", READ },
21   { "begin", BEGIN },
22   { "write", WRITE },
23   { "integer", INTEGER },
24   { "program", PROGRAM },
25   { "writeln", WRITELN },
26   { "variable", VARIABLE } };
27
28 //FUNCTION PROTOTYPES
    ============================================================
29 void syntaxError(int errorType);
30 char mostRecentToken[50] = "";
31 char subtoken;
32 //SCANNER
    ============================================================
   ============
33 // Go through text until a token is found, in which case a
   token
34 // will be returned.
35 token scanner ()
36 {
37     //"Reset" the most recent token so that we can populate
    it again.
38     strcpy(mostRecentToken, "");
39     //Continue to take in characters until we can discern
   what the token is.
40     //feof(src_in)
41     while(!feof(src_in))
42     {
43         subtoken = fgetc(src_in);
```

```
44          strcat(mostRecentToken, &subtoken);
45          if(isalpha(subtoken)) // Token is an ID or a
   RESERVED WORD.
46          {
47              subtoken = fgetc(src_in);
48              //This must be a keyword or id and we need to
   test accordingly.
49              while(isalpha(subtoken) || isdigit(subtoken) ||
    subtoken == '_')
50              {
51                  strcat(mostRecentToken, &subtoken);
52                  subtoken = fgetc(src_in);
53              }
54              //Since we have reached a subtoken that is
   apparently part of the next token, let's put it
55              //back onto src_in for later analysis.
56              ungetc(subtoken, src_in);
57              //Now, we simply have to test whether the token
    matches any reserved words.
58              for(int i = 0; i < sizeof(res_words)/sizeof(
   struct reserved); i++)
59              {
60                  //If a match is found, then return the
   appropriate token value from that reserved word entry.
61                  if(!strcmp(mostRecentToken, res_words[i].
   name))
62                  {
63                      return res_words[i].value;
64                  }
65              }
66              //If a match is not found, then this must be an
    ID.
67              return ID;
68          }else if(isdigit(subtoken)) //The token is an
   integer CONST.
69          {
70              //We must gather the remaining digits and
   return.
71              subtoken = fgetc(src_in);
72              while(isdigit(subtoken))
73              {
74                  strcat(mostRecentToken, &subtoken);
75                  subtoken = fgetc(src_in);
76              }
77              //Eventually, we will obtain a subtoken that is
    not an integer CONST, in which case we must
78              //put that subtoken back onto src_in because it
    is part of the next token (or perhaps
79              //the end of the file has been reached).
80              ungetc(subtoken, src_in);
```

```
 81              return CONST;
 82          }else
 83          {
 84              //Now we can test for any number of cases that
     can be recognized with upon
 85              //seeing the first character of the token.
    Indeed, many of these consist
 86              //only of one character to begin with...
 87              switch(subtoken)
 88              {
 89                  //If this is just whitespace, we can
    ignore it and move onto the next token
 90                  //or acknowledge that the end of the file
    has been reached.
 91          case ' ': strcpy(mostRecentToken, "");
 92                  break;
 93                  case '\n': strcpy(mostRecentToken, "");
 94                  break;
 95                  //The token is ASSIGN. Check to ensure
    that it was entered properly ("<--").
 96                  //Otherwise, report an incomplete
    assignment and move on.
 97                  case '<': subtoken = fgetc(src_in);
 98                                      if(subtoken == '-')
 99                                        {
100                                            subtoken = fgetc(
    src_in);
101                                            if(subtoken == '-'
    )
102                                            {
103                                                //The token
    was entered properly as an ASSIGN operator.
104                                                return ASSIGN;
105                                            }else
106                                            {
107                                                ungetc(
    subtoken, src_in);
108                                                syntaxError(1)
    ;
109                                                break;
110                                            }
111                                        }else
112                                        {
113                                            ungetc(subtoken,
    src_in);
114                                            syntaxError(1);
115                                            break;
116                                        }
117              //The token should be a STRING. Record the
    string until another double quote is encountered.
```

```
118             case '\"': subtoken = fgetc(src_in);
119                                     while(subtoken != '\"'
     && !feof(src_in))
120                                     {
121                                         strcat(
    mostRecentToken, &subtoken);
122                                         subtoken = fgetc(
    src_in);
123                                     }
124                                     //Eventually, another
    double quote should be found. But to be safe,
125                                     //make sure that we
    didn't simply reach the end of the file before
126                                     //another double quote
     was found. In the event that this happens,
127                                     //an error handler
    will print a message before scanning is allowed to
128                                     //proceed normally.
129                                     if(feof(src_in))
130                                     {
131                                         syntaxError(2);
132                                         break;
133                                     }else //Otherwise, the
     token is indeed a STRING.
134                                     {
135                                         strcat(
    mostRecentToken, &subtoken);
136                                         return STRING;
137                                     }
138             //The remaining valid one character tokens.
    Fairly self-expanatory.
139             case '+': return PLUS;
140             case '-': return MINUS;
141             case '(': return LPAREN;
142             case ')': return RPAREN;
143             case ';': return SEMI;
144             case ',': return COMMA;
145             case ':': return COLON;
146             case '.': return PERIOD;
147             //This case is here to address an issue with
    how the End of File is detected. After the last token
148             //in an ATL/0 file is scanned, the while loop
     actually allows fgetc(src_in) to attempt to extract one
    more character,
149             //because feof(src_in) does not detect that
    the End of File has been reached until at least one failed
     attempt
150             //to extract another character has occured.
    The issue with this happening is that fgetc's failed
    attempt returns
```

```
151             //the value of EOF which is obviously not a
    valid character for use in ATL/0: This triggers a
    syntaxError.
152             //This case "catches" that error and ensures
    that the failed attempt is ignored, and the scan proceeds
    to finish gracefully.
153             case EOF: break;
154             //If none of these cases match, then the
    character is illegal. Discard it and proceed.
155             default: syntaxError(3);
156         }
157     }
158  }
159  //If the while loop has terminated, then it must be the
    case that we've reached the end of the file.
160  return SCANEOF;
161 }
162
163 void syntaxError(int errorType)
164 {
165     switch(errorType)
166     {
167         case 1: printf("SYNTAX ERROR 1 DETECTED: An
    incomplete assignment operator was found. Operator
    discarded.\n");
168         case 2: printf("SYNTAX ERROR 2 DETECTED: End of
    file was reached before second double quote was found.\n")
    ;
169         case 3: printf("SYNTAX ERROR 3 DETECTED: Use of
    this character is illegal in ATL/0. Character %s discarded
    .\n", mostRecentToken);
170     }
171 }
172
```