```cpp
#include <string>
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

enum statetype{ newtoken, resword, variable, integer, real, statedelimiter, laststate};
enum chartype{ letter, digit, period, chardelimiter, blank, pod, eoln, illegal, lastchartype};

const int maxstring = 17;

statetype stringtostatetype(string s){
    statetype answer;
    if(s=="NewToken")
        answer = newtoken;
    else if(s=="ReservedWord")
        answer = resword;
    else if(s=="Variable")
        answer = variable;
    else if(s=="Integer")
        answer = integer;
    else if(s=="Real")
        answer = real;
    else if(s=="Delimiter")
        answer = statedelimiter;
    else
        answer = laststate;
    return answer;
}

string statetypetostring(statetype s){
    string answer;
    if(s==newtoken)
        answer = "NewToken";
    else if(s==resword)
        answer = "ReservedWord";
    else if(s==variable)
        answer = "Variable";
    else if(s==integer)
        answer = "Integer";
    else if(s==real)
        answer = "Real";
    else if(s==statedelimiter)
        answer = "Delimiter";
    else
        answer = "Not Valid";
    return answer;
}

chartype stringtochartype(string s){
    chartype answer;
    if(s=="Letter")
        answer = letter;
```

```
55      else if(s=="Digit")
56          answer = digit;
57      else if(s=="Period")
58          answer = period;
59      else if(s=="Delimiter")
60          answer = chardelimiter;
61      else if(s=="Blank")
62          answer = blank;
63      else if(s=="Pod")
64          answer = pod;
65      else if(s=="EOLN")
66          answer = eoln;
67      else if(s=="Illegal")
68          answer = illegal;
69      else
70          answer = lastchartype;
71      return answer;
72  }
73
74  string chartypetostring(chartype s){
75      string answer;
76      if(s==letter)
77          answer = "Letter";
78      else if(s==digit)
79          answer = "Digit";
80      else if(s==period)
81          answer = "Period";
82      else if(s==chardelimiter)
83          answer = "Delimiter";
84      else if(s==blank)
85          answer = "Blank";
86      else if(s==pod)
87          answer = "Pod";
88      else if(s==eoln)
89          answer = "EOLN";
90      else if(s==illegal)
91          answer = "Illegal";
92      else
93          answer = "Not Valid";
94      return answer;
95  }
96
97  template <class Bryn>
98  void swapme(Bryn &first, Bryn &second) {
99  Bryn temp = first;
100 first = second;
101 second = temp;
102 }
103
104 void AlphaBubSort(string array[]) {
105 for (int y = 0; y < maxstring-1; y++){
106     for (int b = 0; b < maxstring - 1; b++) {
107         if (array[b] > array[b+1] && array[b+1] != "") { swapme(array[b],
        array[b+1]); } } }
108 }
109
```

```
110 chartype getchartype(char ch){
111 chartype answ;
112 if (ch >= 'A' && ch <= 'Z') answ = letter;
113 else if (ch >= 'a' && ch <= 'z') answ = letter;
114 else if (ch >= '0' && ch<='9') answ = digit;
115 else if (ch == '$' || ch=='%') answ = pod;
116 else if (ch == '.') answ = period;
117 else if (ch == ' ') answ = blank;
118 else if (ch == '@') answ = eoln;
119 else if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '(' || ch == ')'
    || ch=='=') answ = chardelimiter;
120 else if (ch == ',' || ch == '^' || ch == '"' || ch == '&' || ch == '>' || ch ==
    '<') answ = chardelimiter;
121 else answ = illegal;
122 return answ;
123 }
124
125 void readreserved(string reserves[]){
126 ifstream resinf;
127 resinf.open("reserve.dat");
128 for (int i=0; i<maxstring;i++) {
129     resinf >> reserves[i] >> ws;}
130 }
131
132 void writereserved(string reserves[], ofstream &outf){
133 AlphaBubSort(reserves);
134 outf << setw(5) << right << " " << "Reserved Words Table" << endl;
135 for (int i =0; i < maxstring; i++) {outf << reserves[i]<< endl;}
136 }
137
138 void readprog1(string ProgString[], int &numprog){
139 ifstream inf;
140 inf.open("prog1.bas");
141 numprog=0;
142 while (!inf.eof()) {
143     for (int k = 0; k < maxstring; k++){
144         numprog++;
145         getline(inf,ProgString[k]); } }
146 }
147
148 void writeprog1(string ProgString[], ofstream &outf, int numprog){
149 outf << setw(7) << right << " " << "Prog1.bas Table" << endl;
150 for (int j=0;j<numprog;j++) outf << ProgString[j]<< endl;
151 }
152
153 void readaction(int Action[laststate][lastchartype]){
154     ifstream inf;
155     inf.open("action.dat");
156     for (int i= newtoken;i< laststate; i++)
157         for (int j=letter; j<lastchartype;j++)
158             inf >> Action[i][j];
159 }
160
161 void writeaction(ofstream &outf, int Action[laststate][lastchartype]){
162     outf << endl << endl;
163     outf << setw(50) << "Action Table" << endl;
```

```
164      outf << setw(13) << " ";
165      for (int i=letter;i<lastchartype; i++) { outf << left << setw(8) <<
         chartypetostring((chartype)i) << " "; }
166      outf << endl;
167      for (int k= newtoken;k< laststate; k++) {
168          outf << left << setw(16) << statetypetostring((statetype)k);
169          for (int j=letter; j<lastchartype;j++){ outf << setw(9) << Action[k][j]; }
170           outf << endl;  }
171      outf << endl;
172  }
173
174  void readexplain(string ExplainString[], int &numexplain){
175  ifstream inf;
176  inf.open("explain.dat");
177  numexplain = 0;
178  while (!inf.eof()) {
179      for (int y = 0; y < maxstring; y++){
180           getline(inf,ExplainString[y]);
181           numexplain++; }}
182  }
183
184  void writeexplain(string ExplainString[], ofstream &outf, int numexplain){
185      outf << endl << endl;
186      outf << setw(25) << right << " " << "Explanations Table" << endl;
187      for (int m=0;m<numexplain;m++) outf << ExplainString[m]<< endl;
188  }
189
190  void readstate(statetype FSM[laststate][lastchartype]){
191  string str;
192  ifstream inf;
193  inf.open("state.dat");
194  for (int k= newtoken;k< laststate; k++) {
195      for (int j=letter; j<lastchartype;j++){
196          inf >> str >> ws;
197          FSM[k][j]=stringtostatetype(str); } }
198  }
199
200  void writestate(ofstream &outf, statetype FSM[laststate][lastchartype]){
201  outf << setw(65) << "State Table" << endl;
202  outf << setw(12) << " " ;
203  for (int i=letter;i<lastchartype; i++) {
204      outf << left << setw(13) << "|"+chartypetostring((chartype)i);}
205      outf << endl;
206      outf << right << setw(117) << setfill('-') << " ";
207      outf << setfill(' ') << endl;
208  for (int k= newtoken;k< laststate; k++) {
209      outf << left << setw(12) << statetypetostring((statetype)k) << "|";
210      for (int j=letter; j<lastchartype;j++){
211          outf << left << setw(12) << statetypetostring(FSM[k][j]) << "|"; }
212          outf << endl; }
213  }
214
215  void printtoken(string token, statetype state, ofstream &outf){
216  outf << right << setw(22) << token;
217  outf << setw(15) << statetypetostring(state) << endl;
218  }
```

```
219
220 void searchreserves(string reserves[], string token, statetype &state){
221 bool found = false;
222     for(int i=0; i<maxstring; i++){
223         if (token == reserves[i])
224             found = true; }
225 if (found == false) state=variable;
226 }
227
228 void doactions(char ch, string &token, statetype &state, chartype cct, ofstream
    &outf, int actiontodo, string reserves[]){
229 if (actiontodo==1) {
230     token += ch; }
231 else if (actiontodo==2) {
232     searchreserves(reserves, token, state);
233     printtoken(token, state, outf);
234     token = ""; }
235 else if (actiontodo==3) {
236     printtoken(token, state, outf);
237     token = ""; }
238 else if (actiontodo==4) {
239     printtoken(token, state, outf);
240     outf << "Improper Usage:";
241     outf << setw(7) << ch;
242     outf << endl;
243     token = ""; }
244 else if (actiontodo==5) {
245     outf << "Improper Usage:";
246     outf << setw(7) << ch;
247     outf << endl;}
248 else if (actiontodo==6) {/*Continue*/}
249 else if (actiontodo==7) {
250     outf << "Illegal Character:";
251     outf << setw(4) << ch;
252     outf << endl;}
253 else if (actiontodo==8) {
254     searchreserves(reserves, token, state);
255     printtoken(token, state, outf);
256     token="";
257     token+=ch;}
258 else if (actiontodo==9) {
259     printtoken(token, state, outf);
260     token = "";
261     token += ch; }
262 else if (actiontodo==10) {
263     token += ch;
264     state = variable;
265     printtoken(token, state, outf);
266     token = "";
267 }
268 else if (actiontodo==11) {
269     searchreserves(reserves, token, state);
270     printtoken(token,state,outf);
271     outf <<"Illegal Character:";
272     outf << setw(4) << ch;
273     outf << endl;}
```

```
274  else if (actiontodo==12){
275      token += ch;
276      printtoken(token,state,outf);
277      token="";}
278  else if (actiontodo==13){
279      printtoken(token,state,outf);
280      outf << "Illegal Character:";
281      outf << setw(4) << ch;
282      outf << endl;
283      token = "";}
284  else cout << "ERROR in ActionToDo";
285  }
286
287  void scanner(ofstream &outf, string reserves[],statetype
     FSM[laststate][lastchartype],int Action[laststate][lastchartype] ){
288  ifstream inf;
289  inf.open("prog1.bas");
290  string line;
291  char ch;
292  chartype cct;
293  statetype state = newtoken;
294  string token;
295  outf << endl << endl;
296  outf << right << setw(11) <<" " << "SCANNER RESULTS:" << endl;
297  outf << right <<"Error?";
298  outf << right << setw(16) << "TOKEN";
299  outf << right << setw(15) << "TOKEN-TYPE" << endl;
300  while (!inf.eof()) {
301      getline(inf,line);
302      line += '@';
303      int actiontodo;
304      int length = line.length();
305      for (int i=0; i<length;i++){
306          ch = line[i];
307          cct = getchartype(ch);
308          actiontodo = Action[state][cct];
309          doactions(ch, token, state, cct, outf, actiontodo, reserves);
310          state = FSM[state][cct]; } }
311  }
312
313  int main() {
314  int Action[laststate][lastchartype];
315  statetype FSM[laststate][lastchartype];
316  int numprog;
317  int numexplain;
318
319  ofstream outf;
320  outf.open("outputfile.txt");
321
322  string reserves[maxstring];
323  string ProgString[maxstring];
324  string ExplainString[maxstring];
325
326  readreserved(reserves);
327  writereserved(reserves, outf);
328
```

```
329   readprog1(ProgString, numprog);
330   writeprog1(ProgString, outf, numprog);
331
332   readaction(Action);
333   writeaction(outf, Action);
334
335   readexplain(ExplainString, numexplain);
336   writeexplain(ExplainString, outf, numexplain);
337
338   readstate(FSM);
339   writestate(outf,FSM);
340
341   scanner(outf, reserves,FSM, Action);
342
343   system("pause");
344   }
345
```