```c
#include<stdio.h>
#include<string.h>
#define SIZE(x) (sizeof(x)/sizeof(x[0]))
//structure for each indivisual token
typedef struct stud1{
char name[100];
char type[100];
int occurences;
int count;
int id;
int line_no[100][50];
}token;
//Symbol Table
typedef struct stud2{
token t[10000];
int n;
}symbol_table;
symbol_table st;
//to create a structure for each valid token in our language
void verify(char *check,int k,char *w)
{
    strcpy(st.t[k].name,check);
    strcpy(st.t[k].type,w);
    st.t[k].occurences=0;
    st.t[k].line_no[0][0]=0;
    st.t[k].count=1;
    st.t[k].id=st.n;
    st.n++;

}
//Declaring all various tokens in our defined languge
void valid_tokens()
{
    char keyword[][20]={"auto","extern","sizeof","break","float","static","case",
                    "for","struct","char","goto","switch","const","if","typedef",
                    "continue","int","union","default","long","unsigned","do",
                    "register","void","return","double","volatile","else","short",
                    "while","enum","signed"};
    char arithmetic[][20]={"+","-","*","/","%"};
    char relational[][20]={"<",">",">=","<=","==","!="};
    char conditional[][20]={"&&","||","!"};
    char assignment[][20]={"="};
    char inc_dec[][20]={"++","--"};
    char punctuation[][20]={"{",",",";","}"," "};
    int k=0,i;
   //keyword token
    for(i=0;i<SIZE(keyword);i++)
    {
      verify(keyword[i],k,"KEYWORD");
      k++;
    }
    //arithmetic token
    for(i=0;i<SIZE(arithmetic);i++)
    {
    verify(arithmetic[i],k,"ARITHMETIC OPERATOR");
      k++;
    }
```

```
58        //relational token
59        for(i=0;i<SIZE(relational);i++)
60        {
61        verify(relational[i],k,"RELATIONAL OPERATOR");
62          k++;
63        }
64        //conditional token
65        for(i=0;i<SIZE(conditional);i++)
66        {
67        verify(conditional[i],k,"CONDITIONAL OPERATOR");
68          k++;
69        }
70        //assignment token
71        for(i=0;i<SIZE(assignment);i++)
72        {
73        verify(assignment[i],k,"ASSIGNMENT OPERATOR");
74          k++;
75        }
76        //Increment-decrement token
77        for(i=0;i<SIZE(inc_dec);i++)
78        {
79        verify(inc_dec[i],k,"INC-DEC OPERATOR");
80          k++;
81        }
82        //Puctuation token
83        for(i=0;i<SIZE(punctuation);i++)
84        {
85        verify(punctuation[i],k,"PUNCTUATION SYMBOL");
86        k++;
87        }
88    }
89    void check(char* w,int line,int pos)
90    {
91        int i,flag=0;
92        //compare this word with all our existing tokens
93        for(i=0;i<st.n;i++)
94        {
95          if(!strcmp(w,st.t[i].name))
96               break;
97        }
98        //Create a structure for this token if this word is not present in our defined ↵
          set of tokens
99        if(i!=st.n)
100       {
101           if(st.t[i].line_no[st.t[i].occurences][0]==line)
102           {
103               if(st.t[i].occurences==0)
104                   st.t[i].occurences++;
105               st.t[i].line_no[st.t[i].occurences][st.t[i].count]=pos;
106               st.t[i].count++;
107           }
108           else
109           {
110           st.t[i].occurences++;
111           st.t[i].line_no[st.t[i].occurences][0]=line;
112           st.t[i].count=1;
113           st.t[i].line_no[st.t[i].occurences][st.t[i].count]=pos;
```

```
114           st.t[i].count=2;
115           }
116       }
117       else
118       {
119         if(!(w[0]>='0' && w[0]<='9'))
120          {
121             //check for a Valid Identifier
122           for(i=0;i<strlen(w);i++)
123            {
124            if((w[i]>='A' && w[i]<='Z')||(w[i]>='a' && w[i]<='z')||(w[i]=='_')||(w[i
           ]>='0' && w[i]<='9'))
125               flag=1;
126            else
127            {
128                flag=0;
129                break;
130            }
131           }
132           //Valid Token Identifier
133           if(flag)
134           {
135            if(st.t[st.n].line_no[1][0]==line)
136             {
137              st.t[st.n].line_no[0][st.t[st.n].count]=pos;
138              st.t[st.n].count++;
139            }
140            else
141            {
142            strcpy(st.t[st.n].name,w);
143            strcpy(st.t[st.n].type,"IDENTIFIER");
144            st.t[st.n].line_no[1][0]=line;
145            st.t[st.n].count=1;
146            st.t[st.n].line_no[1][st.t[st.n].count]=pos;
147            st.t[st.n].count=2;
148            st.t[st.n].occurences=1;
149            st.t[st.n].id=st.n;
150            st.n++;
151            }
152          }
153         }
154         //Invalid Token Reporting ERROR
155         if (!flag)
156         {
157            if(st.t[st.n].line_no[1][0]==line)
158             {
159              st.t[st.n].line_no[1][st.t[st.n].count]=pos;
160              st.t[st.n].count++;
161            }
162            else
163            {
164            strcpy(st.t[st.n].name,w);
165            strcpy(st.t[st.n].type,"INVALID");
166            st.t[st.n].line_no[1][0]=line;
167            st.t[st.n].count=1;
168            st.t[st.n].line_no[1][st.t[st.n].count]=pos;
169            st.t[st.n].count=2;
```

```
170             st.t[st.n].occurences=1;
171             st.t[st.n].id=st.n;
172             st.n++;
173             }
174         }
175       }
176   }
177   int other(char c)
178   {
179     //Valid Symbols in our language other than alphanumeric characters
180     char valid[]={'+','-','*','/','%','>','<','=','!','&','|'};
181     int i;
182     for(i=0;i<sizeof(valid);i++)
183     {
184         if(valid[i]==c)
185           return 1;
186     }
187     return 0;
188   }
189   int main()
190   {
191       int line=1,i,q=0,j,w,prev,pos=1;
192       char filename[100],s[100000],word[100000];
193       st.n=0;
194       printf("Enter the file name with.txt extension\n");
195       scanf("%s", filename);
196       FILE *fp=fopen(filename, "r");
197       //Read file line by line
198       valid_tokens();
199       while(fgets(s,sizeof(s),fp)!=NULL){
200               int flag=0;
201               q=0;
202         for(i=0;i<strlen(s);i++)
203          {
204              //Check for an alphanumeric character
205              if(!flag && ((s[i]>='A' && s[i]<='Z')||(s[i]>='a' && s[i]<='z')||(s[i
                 ]>='0' && s[i]<='9')||(s[i]=='_')))
206                   {
207                      word[q++]=s[i];
208                   }
209              else
210                  {
211                    //For symbols like ++,--,<=,>= to be valid
212                   if(flag && other(s[i]))
213                        {
214                          word[q++]=s[i];
215                        }
216                     else
217                     {
218                       //Not read tab character
219                      if(s[i]!='\t')
220                       {
221                          //Not read Null character
222                          if(q!=0)
223                            {
224                              word[q]='\0';
225                              check(word,line,pos);
```

```
226                             pos++;
227                          }
228                       }
229                      q=0;
230                     // All characters accepted other than tab ,carriage return and
                        new line
231                      if(s[i]!='\t' && s[i]!='\n')
232                       {
233                        word[q++]=s[i];
234                        //Special characters
235                        if(!(other(s[i])))
236                           {
237                               //Present alphabet is an alphanumeric character
238                               if((s[i]>='A' && s[i]<='Z')||(s[i]>='a' && s[i]<='z'
                               )||(s[i]>='0' && s[i]<='9'))
239                                 {
240                                   flag=0;
241                                 }
242                                 else
243                                 {
244                               //Check if the word is a valid token
245                                 word[q]='\0';
246                                 check(word,line,pos);
247                                 pos++;
248                                 q=0;
249                                 flag=0;
250                                 }
251                           }
252                        else
253                          flag=1;
254                       }
255                       if(s[i]=='\n')
256                          pos=1;
257                     }
258                }
259          }
260         //Increment line number
261         line++;
262       }
263           //Print Symbol Table
264       printf("\n\t\t\t\tSymbol Table\n");
265       printf("\n\tID\tTOKEN NAME  \tTOKEN TYPE \t\tOCCURENCES\tLINE
          NUMBERS(POSITION NUMBER)\n");
266       for(i=0;i<(st.n);i++)
267       {
268           if(st.t[i].occurences>=1)
269           {
270               printf("\n  %3d\t%20s\t%20s\t%5d\t\t",st.t[i].id,st.t[i].name,st.t[i].
                  type,st.t[i].occurences);
271               prev=-1;
272               for(j=1;j<=st.t[i].occurences;j++)
273                  {
274                    if(!(prev==st.t[i].line_no[j][0]))
275                    {
276                      printf("%d (",st.t[i].line_no[j][0]);
277                      for(w=1; w<10 && st.t[i].line_no[j][w]!=0;w++)
278                      printf("%d ",st.t[i].line_no[j][w]);
```

```
279                     printf("),");
280                     prev=st.t[i].line_no[j][0];
281                   }
282                 }
283               printf("\n");
284           }
285       }
286       return 0;
287   }
288
289
```