

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<string.h>
4  #define SIZE(x) (sizeof(x)/sizeof(x[0]))
5  //structure for each indivisual token
6  typedef struct stud1{
7  char name[100];
8  char type[100];
9  int occurences;
10 int line_no[10000];
11 }token;
12 //Symbol Table
13 typedef struct stud2{
14 token t[10000];
15 int n;
16 }symbol_table;
17 symbol_table st;
18 //to create a structure for each valid token in our language
19 void verify(char *check,int k,char *w)
20 {
21     strcpy(st.t[k].name,check);
22     strcpy(st.t[k].type,w);
23     st.t[k].occurences=0;
24     st.t[k].line_no[0]=0;
25     st.n++;
26 }
27 //Declaring all various tokens in our defined language
28 void valid_tokens()
29 {
30     char keyword[][20]={"auto","extern","sizeof","break","float","static","case",
31                         "for","struct","char","goto","switch","const","if","typedef",
32                         "continue","int","union","default","long","unsigned","do",
33                         "register","void","return","double","volatile","else","short",
34                         "while","enum","signed"};
35     char arithmetic[][20]={"+","-","*","/","%"};
36     char relational[][20]={"<",">",">=","<=","==","!="};
37     char conditional[][20]={"&&","||","!"};
38     char assignment[][20]={"="};
39     char inc_dec[][20]={"++","--"};
40     char punctuation[][20]={"{","(",",",";","}",","," "};
41     int k=0,i;
42     //keyword token
43     for(i=0;i<SIZE(keyword);i++)
44     {
45         verify(keyword[i],k,"KEYWORD");
46         k++;
47     }
48     //arithmetic token
49     for(i=0;i<SIZE(arithmetic);i++)
50     {
51         verify(arithmetic[i],k,"ARITHMETIC OPERATOR");
52         k++;
53     }
54     //relational token
55     for(i=0;i<SIZE(relational);i++)
56     {
57         verify(relational[i],k,"RELATIONAL OPERATOR");

```

```

58     k++;
59 }
60 //conditional token
61 for(i=0;i<SIZE(conditional);i++)
62 {
63     verify(conditional[i],k,"CONDITIONAL OPERATOR");
64     k++;
65 }
66 //assignment token
67 for(i=0;i<SIZE(assignment);i++)
68 {
69     verify(assignment[i],k,"ASSIGNMENT OPERATOR");
70     k++;
71 }
72 //Increment-decrement token
73 for(i=0;i<SIZE(inc_dec);i++)
74 {
75     verify(inc_dec[i],k,"INC-DEC OPERATOR");
76     k++;
77 }
78 //Punctuation token
79 for(i=0;i<SIZE(punctuation);i++)
80 {
81     verify(punctuation[i],k,"PUNCTUATION SYMBOL");
82     k++;
83 }
84 }
85 void check(char* w,int line)
86 {
87     int i,flag=0;
88     //compare this word with all our existing tokens
89     for(i=0;i<st.n;i++)
90     {
91         if(!strcmp(w,st.t[i].name))
92             break;
93     }
94     //Create a structure for this token if this word is not present in our defined
    set of tokens
95     if(i!=st.n)
96     {
97         st.t[i].line_no[st.t[i].occurences]=line;
98         st.t[i].occurences++;
99     }
100     else
101     {
102         if(!(w[0]>='0' && w[0]<='9'))
103         {
104             //check for a Valid Identifier
105             for(i=0;i<strlen(w);i++)
106             {
107                 if((w[i]>='A' && w[i]<='Z') || (w[i]>='a' && w[i]<='z') || (w[i]=='_') || (w[i]
                 ]>='0' && w[i]<='9'))
108                     flag=1;
109                 else
110                 {
111                     flag=0;
112                     break;

```

```

113     }
114 }
115 //Valid Token Identifier
116 if(flag)
117 {
118     strcpy(st.t[st.n].name,w);
119     strcpy(st.t[st.n].type,"IDENTIFIER");
120     st.t[st.n].line_no[0]=line;
121     st.t[st.n].occurences=1;
122     st.n++;
123 }
124 }
125 //Invalid Token Reporting ERROR
126 if (!flag)
127 {
128     strcpy(st.t[st.n].name,w);
129     strcpy(st.t[st.n].type,"INVALID");
130     st.t[st.n].line_no[0]=line;
131     st.t[st.n].occurences=1;
132     st.n++;
133 }
134 }
135 }
136 int other(char c)
137 {
138     //Valid Symbols in our language other than alphanumeric characters
139     char valid[]={'+', '-', '*', '/', '%', '>', '<', '=', '!', '&', '|'};
140     int i;
141     for(i=0;i<sizeof(valid);i++)
142     {
143         if(valid[i]==c)
144             return 1;
145     }
146     return 0;
147 }
148 int main()
149 {
150     int line=1,i,q=0,j,prev;
151     char filename[100],s[100000],word[100000];
152     st.n=0;
153     printf("Enter the file name with.txt extension\n");
154     scanf("%s", filename);
155     FILE *fp=fopen(filename, "r");
156     if(fp==NULL)
157     {
158         printf("Error File Cannot be opened\n");
159         exit(0);
160     }
161     //Read file line by line
162     valid_tokens();
163     while(fgets(s,sizeof(s),fp)!=NULL){
164         int flag=0;
165         q=0;
166         for(i=0;i<strlen(s);i++)
167         {
168             //Check for an alphanumeric character
169             if(!flag && ((s[i]>='A' && s[i]<='Z') || (s[i]>='a' && s[i]<='z')) || (s[i]>= '

```

```

170         '0' && s[i]<='9')||(s[i]=='_'))
171     {
172         word[q++]=s[i];
173     }
174     else
175     {
176         //For symbols like ++,--,<=,>= to be valid
177         if(flag && other(s[i]))
178         {
179             word[q++]=s[i];
180         }
181         else
182         {
183             //Not read tab character
184             if(s[i]!='\t')
185             {
186                 //Not read Null character
187                 if(q!=0)
188                 {
189                     word[q]='\0';
190                     check(word,line);
191                 }
192                 q=0;
193                 // All characters accepted other than tab ,carriage return and
194                 // new line
195                 if(s[i]!='\t' && s[i]!='\n' && s[i]!='\r')
196                 {
197                     word[q++]=s[i];
198                     //Special characters
199                     if(!(other(s[i])))
200                     {
201                         //Present alphabet is an alphanumeric character
202                         if((s[i]>='A' && s[i]<='Z')||(s[i]>='a' && s[i]<='z'
203                         )||(s[i]>='0' && s[i]<='9'))
204                         {
205                             flag=0;
206                         }
207                         else
208                         {
209                             //Check if the word is a valid token
210                             word[q]='\0';
211                             check(word,line);
212                             q=0;
213                             flag=0;
214                         }
215                     }
216                     else
217                     {
218                         flag=1;
219                     }
220                 }
221             }
222         }
223         //Increment line number
224         line++;
225     }
226     //Print Symbol Table

```

```
224     printf("\n\t\t\t\t\tSymbol Table\n");
225     printf("\n\tTOKEN NAME \tTOKEN TYPE \t\tOCCURENCES\tLINE NUMBERS\n");
226     for(i=0;i<(st.n);i++)
227     {
228         if(st.t[i].occurences>=1)
229         {
230             //if the token has occurred then print it,its type, no of occurences and line number
231             printf("\n%20s\t%20s\t%5d\t%10d",st.t[i].name,st.t[i].type,st.t[i].occurences,st.t[i].line_no[0]);
232             prev=st.t[i].line_no[0];
233             for(j=1;j<st.t[i].occurences;j++)
234             {
235                 if(!(prev==st.t[i].line_no[j]))
236                 {
237                     printf(",%d",st.t[i].line_no[j]);
238                     prev=st.t[i].line_no[j];
239                 }
240             }
241             printf("\n");
242         }
243     }
244     return 0;
245 }
246
247
```