```c
#include<stdio.h>
#include<stdlib.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 line_no[10000];
}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;
    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++;
    }
    //relational token
    for(i=0;i<SIZE(relational);i++)
    {
    verify(relational[i],k,"RELATIONAL OPERATOR");
```

```
          k++;
        }
        //conditional token
        for(i=0;i<SIZE(conditional);i++)
        {
        verify(conditional[i],k,"CONDITIONAL OPERATOR");
          k++;
        }
        //assignment token
        for(i=0;i<SIZE(assignment);i++)
        {
        verify(assignment[i],k,"ASSIGNMENT OPERATOR");
          k++;
        }
        //Increment-decrement token
        for(i=0;i<SIZE(inc_dec);i++)
        {
        verify(inc_dec[i],k,"INC-DEC OPERATOR");
          k++;
        }
        //Puctuation token
        for(i=0;i<SIZE(punctuation);i++)
        {
        verify(punctuation[i],k,"PUNCTUATION SYMBOL");
        k++;
        }
    }
    void check(char* w,int line)
    {
        int i,flag=0;
        //compare this word with all our existing tokens
        for(i=0;i<st.n;i++)
        {
          if(!strcmp(w,st.t[i].name))
             break;
        }
        //Create a structure for this token if this word is not present in our defined set of tokens
        if(i!=st.n)
        {
            st.t[i].line_no[st.t[i].occurences]=line;
            st.t[i].occurences++;
        }
        else
        {
          if(!(w[0]>='0' && w[0]<='9'))
           {
              //check for a Valid Identifier
            for(i=0;i<strlen(w);i++)
             {
              if((w[i]>='A' && w[i]<='Z')||(w[i]>='a' && w[i]<='z')||(w[i]=='_')||(w[i]>='0' && w[i]<='9'))
                 flag=1;
              else
              {
                  flag=0;
                  break;
```

```c
              }
            }
            //Valid Token Identifier
            if(flag)
            {
              strcpy(st.t[st.n].name,w);
              strcpy(st.t[st.n].type,"IDENTIFIER");
              st.t[st.n].line_no[0]=line;
              st.t[st.n].occurences=1;
              st.n++;
            }
           }
          //Invalid Token Reporting ERROR
          if (!flag)
          {
              strcpy(st.t[st.n].name,w);
              strcpy(st.t[st.n].type,"INVALID");
              st.t[st.n].line_no[0]=line;
              st.t[st.n].occurences=1;
              st.n++;
          }
        }
    }
    int other(char c)
    {
      //Valid Symbols in our language other than alphanumeric characters
      char valid[]={'+','-','*','/','%','>','<','=','!','&','|'};
      int i;
      for(i=0;i<sizeof(valid);i++)
      {
          if(valid[i]==c)
            return 1;
      }
      return 0;
    }
    int main()
    {
        int line=1,i,q=0,j,prev;
        char filename[100],s[100000],word[100000];
        st.n=0;
        printf("Enter the file name with.txt extension\n");
        scanf("%s", filename);
        FILE *fp=fopen(filename, "r");
        if(fp==NULL)
           {
              printf("Error File Cannot be opened\n");
              exit(0);
           }
        //Read file line by line
        valid_tokens();
        while(fgets(s,sizeof(s),fp)!=NULL){
                int flag=0;
                q=0;
          for(i=0;i<strlen(s);i++)
           {
               //Check for an alphanumeric character
                if(!flag &&((s[i]>='A' && s[i]<='Z')||(s[i]>='a' && s[i]<='z')||(s[i]>= ⮐
              '0' && s[i]<='9')||(s[i]=='_')))
                   {
                      word[q++]=s[i];
                   }
              else
                  {
                  //For symbols like ++,--,<=,>= to be valid
                     if(flag && other(s[i]))
                        {
                          word[q++]=s[i];
                        }
                     else
                     {
                       //Not read tab character
                      if(s[i]!='\t')
                       {
                           //Not read Null character
                           if(q!=0)
                            {
                              word[q]='\0';
                              check(word,line);
                            }
                       }
                      q=0;
                      // All characters accepted other than tab ,carriage return and new line
                      if(s[i]!='\t' && s[i]!='\n' && s[i]!='\r')
                       {
                        word[q++]=s[i];
                        //Special characters
                        if(!(other(s[i])))
                           {
                               //Present alphabet is an alphanumeric character
                                 if((s[i]>='A' && s[i]<='Z')||(s[i]>='a' && s[i]<='z')||(s[i]>='0' && s[i]<='9'))
                                 {
                                   flag=0;
                                 }
                                 else
                                 {
                               //Check if the word is a valid token
                                 word[q]='\0';
                                 check(word,line);
                                 q=0;
                                 flag=0;
                                 }
                           }
                         else
                           flag=1;
                        }
                      }
                  }
          }
          //Increment line number
         line++;
       }
       //Print Symbol Table
```

```c
     printf("\n\t\t\t\tSymbol Table\n");
     printf("\n\tTOKEN NAME  \tTOKEN TYPE \t\tOCCURENCES\tLINE NUMBERS\n");
     for(i=0;i<(st.n);i++)
     {
         if(st.t[i].occurences>=1)
         {
             //if the token has occured then print it,its type, no of occurences and line number
             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]);
             prev=st.t[i].line_no[0];
             for(j=1;j<st.t[i].occurences;j++)
                {
                  if(!(prev==st.t[i].line_no[j]))
                  {
                    printf(",%d",st.t[i].line_no[j]);
                    prev=st.t[i].line_no[j];
                  }
                }
             printf("\n");
         }
     }
     return 0;
}
```