**IMPLEMENTATION OF SYMBOL TABLE**

**Aim:**

To write a C program to implement symbol table

**Description:**

Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities such as variable names, function names, objects, classes, interfaces, etc. Symbol table is used by both the analysis and the synthesis parts of a compiler.

A symbol table may serve the following purposes depending upon the language in hand:

* To store the names of all entities in a structured form at one place.
* To verify if a variable has been declared.
* To implement type checking, by verifying assignments and expressions in the source code are semantically correct.
* To determine the scope of a name

**Algorithm:**

1. Define a Symbol table with following attributes. <symbol name, type, scope, size>
2. Read the source program from the file.
3. If a symbol is read
   1. Check whether the symbol is already in the symbol table using *lookup* operation.
   2. If the symbol is in table then
      1. Report Error – Duplicate Symbol
   3. If not found
      1. Insert the symbol in the symbol table along with its type, scope and size using *Insert* Operation.
4. Write the output (Symbol table) in a file.

**/\*SYMBOL TABLE\*/**

#include <stdio.h>

#include <string.h>

struct symtab

{

char id;

char type[10];

int size;

char scope[25];

};

void main()

{

struct symtab s[15];

int sindex=0,i,flag=0;

char scope[25];

char c,a[100];

FILE \*fptr;

fptr=fopen("sample.txt","r");

strcpy(scope,"global");

while(!feof(fptr))

{

fscanf(fptr,"%s",a);

if(strcmp(a,"main()")==0)

strcpy(scope,"local");

if(!strcmp(a,"int"))

{

fseek(fptr,1,1);

c=fgetc(fptr);

s[sindex].id = c;

strcpy(s[sindex].type,"int");

s[sindex].size = 2;

strcpy(s[sindex].scope,scope);

sindex++;

}

if(!strcmp(a,"float"))

{

fseek(fptr,1,1);

c=fgetc(fptr);

s[sindex].id = c;

strcpy(s[sindex].type,"float");

s[sindex].size = 4;

strcpy(s[sindex].scope,scope);

sindex++;

}

if(!strcmp(a,"char"))

{

fseek(fptr,1,1);

c=fgetc(fptr);

s[sindex].id = c;

strcpy(s[sindex].type,"char");

s[sindex].size = 1;

strcpy(s[sindex].scope,scope); sindex++;

}

}

fclose(fptr);

printf("Name\tType\tSize\tScope\n");

printf("===========================\n");

for(i=0;i<sindex;i++)

printf("%c\t%s\t%d\t%s\n",s[i].id,s[i].type,s[i].size,s[i].scope);

}

**/\*Input: sample.txt\*/**

#include<stdoi.h>

int g;

void main()

{

int a;

float c;

char e;

}

**/\*Ouput\*/**

Name Type Size Scope

==============================

g int 2 global

a int 2 local

c float 4 local

e char 1 local

**Ex.No:**

**LEXICAL ANALYZER USING C**

**Aim:**

To write a C program to implement Lexical Analyzer to identify the patterns like identifiers, keywords, comments, operators and constants.

**Description:**

The first step of a compiler is lexical analysis. It is the process of converting a sequence of characters into a sequence of tokens separated by white spaces and punctuation. A program that performs lexical analysis is calledlexical analyzer.

**Algorithm:**

1. Read the source program from the file.
2. Read and Separate a token using delimiter (whitespace or semicolon).
3. Check the token
   1. If the token starts with “\\” then display the entire line as COMMENTS
   2. If the token starts with “/\*” then display the lines until “\*/” as COMMENTS
   3. If the token starts with “#” then display that as PREPROCESSOR DIRECTIVE
   4. If the token is any keyword like int, if, else, etc.. display that as KEYWORD
   5. If the token is operator (+,-,\*, etc..) then display that as OPERATOR.
   6. If the token (,),{,} then display that as Special Symbol.
   7. If the token is numbers between 0 and 9 then display that as CONSTANT.
   8. Else check whether token starts with alphabet or ‘-‘. If so display that as identifier. Else print invalid identifier.
4. Write the output in a file.

**/\* Lexical Analyzer using C \*/**

#include<stdio.h>

void main()

{

char c,a[100];

FILE \*fptr;

fptr=fopen("sample.txt","r");

while(!feof(fptr))

{

fscanf(fptr,"%s",a);

if(!strcmp(a,"int")||!strcmp(a,"float"))

{

printf("\n%s-keyword",a);

fseek(fptr,1,1);

l:c=fgetc(fptr);

if(c==',')

{

printf("\n%c-others\n",c);

goto l;

}

if(c!=';')

{

if(c=='=')

{

printf("\n%c-operator",c);

c=fgetc(fptr);

printf("\n%c-constant",c);

}

printf("\n%c-identifier",c);

goto l;

}

}

else if(a[0]=='#')

printf("\n%s-preprocessor directive",a);

else if(!strcmp(a,"main()"))

printf("\n%s-identifier",a);

else

printf("\n%s-others\n",a);

}

fclose(fptr);

}

**/\*Input:Sample.txt\*/**

#include<stdio.h>

void main()

{

int a;

float b,c;

}

**/\*Output:\*/**

#include<stdio.h>-preprocessor directive

void-others

main()-identifier

{-others

int-keyword

a-identifier

float-keyword

b-identifier

,-others

c-identifier

}-others