CS6612 – Compiler Lab

Ex no : 2 **Name : Sreedhar V**

Date : 03.02.2021 **Reg no: 185001161**

Specification

Develop a Lexical analyzer to recognize the patterns namely, identifiers, constants, comments and operators using the following regular expressions. Construct symbol table for the identifiers with the following information using LEX tool.

Code

```
%{
#include<stdio.h>
#include<string.h>
int i = 0;
int address=1000;
int size =0;
int flag =1;
char buffer[100];
struct table{
    char symbol[50];
    char type[50];
    int address;
    char value[100];
    int size;
}t[100];
void add_symbol(char a[]);
int lookup(char a[]);
void add_value(char val[],int s);
void display();
void update(char a[]);
%}
/* Rules Section*/
MCMT "/*"([^*]|\*+[^*/])*\*+"/"
ARTHOP [+|-|*|/|^|%]
FC [a-zA-Z]+[(].*[)]
ASSIGN ["="]
RELOP [<|>|!|?|==|<=|>=]
```

```
LOGOP [&&|"||"|"|"|<<|>>|~]
SYM ['{'|'}'|';'|'|'|'.'|':'|')'|'('|,]
INT [-]?[0-9]+
FLOAT [0-9]*"."[0-9]+
ID [a-zA-Z_][a-zA-Z0-9_]*
STR ["][a-zA-Z0-9]["]
SCMT [/][/].*
CHAR ['][a-zA-Z0-9]['] 
%%
return|int|float|long|double|char|if|else {printf("KW ");update(yytext)
;}
{MCMT} {printf("MULTI LINE CMT");}
{FC} {printf("FC ");}
{ASSIGN} {printf("ASSIGN ");flag=1;}
{STR} {printf("STRING ");}
{CHAR} {printf("CHAR ");add_value(yytext,1);address++;}
{SCMT} {printf("SINGLE LINE CMT");continue;}
{ARTHOP} {printf("ARTHOP ");}
{RELOP} {printf("RELOP ");}
{LOGOP} {printf("LOGOP ");}
{SYM} {printf("SYM ");flag=0;}
{FLOAT} {printf("FLOAT ");if(flag)add_value(yytext,4);address+=4;}
{INT} {printf("INT ");if(flag)add_value(yytext,2);address+=2;}
{ID} {printf("ID ");if(lookup(yytext))add_symbol(yytext);}
%%
int yywrap(void){}
int lookup(char a[])
{
    int i=0;
    for(int i=0;i<size;i++)
    {
        if(!strcmp(t[i].symbol,a))
            return 0;
    }
    return 1;
}
void add_value(char val[],int s)
{
    size--;
    strcpy(t[size].value,val);
    t[size].size = s;
    size++;
}
void add_symbol(char a[])
{
    strcpy(t[size].symbol,a);
```

```
    strcpy(t[size].value,"NULL");
    strcpy(t[size].type,buffer);
    t[size].address = address;
    size++;
}
void update(char a[])
{
    strcpy(buffer,a);
}
void display()
{
    int i=0,j;
    printf("\n Starting Address = 1000");
    printf("\n\n SYMBOL TABLE\n");
    printf("\nSYMBOL\tValue\tType  \tAddr\tSize\n");
    for(i=0;i<40;i++)printf("-");
    printf("\n");
    for(i=0;i<size;i++)
    {
        printf("%-6s\t%-5s\t%-6s\t%-
7d\t%d\n",t[i].symbol,t[i].value,t[i].type,t[i].address,t[i].size);
    }
    printf("\n\n");
}

int main()
{
    // The function that starts the analysis
    yyin = fopen("input.c","r");
    yylex();
    display();
    return 0;
}
```

(Sample input file)

```
/*
Multi line comment..
hi
hi
hi
*/
int main()
{
    int a=5;
```

```
    float b=10.13;
    float c;
    if(a>b)
        printf("a_is_greater");
    else
        printf("b is greater");
    add(a,b);
    // This is a comment......
    char out = 'd' > "8" ? a & b : a || b ;
    int var =0, v =9;
}
```

(Output)

```
PS G:\Academics\SSN\6th Sem\Compiler Design\Lex> ./a



MULTI LINE CMT

KW  FC

SYM

    KW  ID ASSIGN INT SYM

    KW  ID ASSIGN FLOAT SYM

    KW  ID SYM

    FC

        FC SYM

    KW

        FC SYM

    FC SYM

    SINGLE LINE CMT

    KW  ID  ASSIGN  CHAR  RELOP  STRING  RELOP  ID  LOGOP  ID  SYM  ID  ARTHOP
ARTHOP  ID  SYM

    KW  ID  ASSIGN INT SYM  ID  ASSIGN INT SYM

SYM
```

```
 Starting Address = 1000

 SYMBOL TABLE

SYMBOL  Value   Type    Addr    Size

---------------------------------------

a       5       int     1000    2

b       10.13   float   1002    4

c       NULL    float   1006    0

out     'd'     char    1006    1

var     0       int     1007    2

v       9       int     1009    2
```

Learning Outcome:

- I’ve learnt how the lexical analyser works and its basic functionalities.
- I’ve learnt how to tokenize an entire C program using lex tool.
- I’ve learnt how to identify and group the lexemes into specific categories.
- I’ve learnt how to construct the symbol table for the identifiers.
- I’ve learnt how to recognize the pattern (regular expression) and separate them into tokens for a program.
- I’ve learnt the regular expression for identifiers, constants, comments and operators.