CS6612 – Compiler Lab

Ex no : 1 **Name : Sreedhar V**

Date : 03.02.2021 **Reg no: 185001161**

Specification

Develop a Lexical analyser to recognize the patterns namely, identifiers, constants, comments and operators using the following regular expressions.

Code

(predefined.h file)

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<limits.h>
#include<stdbool.h>
#include<ctype.h>
const int T = 1;
const int F = 0;
// Creating a global array of keywords , operators, symbols

char keywords[][100]={"return","int","float","long","double","char","if
","else"};
char operators[][10]={"+","-
","*","/","^","%","<",">","!","?","==","<=",">=","||","&&"};
char symbol[]={'{','}',';',',','.',':',')','('};
int sym_size = 9;
int key_size = 10;
int op_size = 15;
int relop_size = 7;
int arthop_size = 6;
char arth_operators[][10]={"+","-","*","/","^","%",};
int logop_size = 4;
char log_operators[][10] = {"||","&&","&","|"};
char rel_operators[][10]={"<",">","!","?","==","<=",">="};
```

(Analyser.c file)

```
#include"predefined.h"


/*Code checks for the following patterns
Identifier
Constant
Comments
Operators
Keywords*/

bool issymbol(char ch[100])
{
    int i;
    for(i=0;i<sym_size;i++)
    {
        if(ch[0]==symbol[i])
            return true;
    }
    return false;
}
bool check_function(char str[100])
{
    int i=0;
    bool a,b;
    for(i=0;i<strlen(str);i++)
    {
        if(str[i]=='(')
            a=true;
        if(str[i]==')')
            b=true;
    }
    if(a && b)
        return true;
    return false;
}
bool check_operator(char str[100])
{
    int i;
    for(i=0;i<op_size;i++)
    {
        if(strcmp(str,operators[i])==0)
            return true;
    }
    return false;
```

```
}
bool check_RELOP(char *str)
{
    int i;
    for(i=0;i<relop_size;i++)
    {
        if(strcmp(str,rel_operators[i])==0)
            return true;
    }
    return false;

}
bool check_LOGOP(char *str)
{
    int i;
    for(i=0;i<logop_size;i++)
    {
        if(strcmp(str,log_operators[i])==0)
            return true;
    }
    return false;

}
bool check_ARTHOP(char *str)
{
    int i;
    for(i=0;i<arthop_size;i++)
    {
        if(strcmp(str,arth_operators[i])==0)
            return true;
    }
    return false;

}
bool check_comment(char *str)
{
    if(str[0]=='\\' || str[1]=='\\')
        return true;
    return false;
}
bool check_keyword(char str[100])
{
    int i;
    for(i=0;i<key_size;i++)
    {
        if(strcmp(str,keywords[i])==0)
```

```
            return true;

    }
    return false;
}
bool check_assign(char ch)
{
    if(ch=='=')
        return true;
    return false;
}
bool check_num(char str[100])
{
    int len = strlen(str);
    int i=0;
    while(i<len)
    {
        if(!isdigit(str[i]))
            return false;
        i++;
    }
    return true;
}
bool check_char(char str[100])
{
    if((str[0]=='\"' || str[0]=='\'' ) && (str[strlen(str)-
1]=='\"' || str[strlen(str)-1]=='\''))
        return true;
    return false;
}

void analyser(char input[100000])
{
    int i=0,temp=0;
    int len = strlen(input);
    char line[100][1000];
    int l=0;
    int flag=0;
    char *token = strtok(input,"\n");
    while(token!=NULL)
    {
        strcpy(line[l++],token);
        token = strtok(NULL,"\n");
    }
    int l1=0;
    while(l1<l)
```

```
{
    if((line[l1][0]=='/') && (line[l1][1]=='/'))
    {
            printf(" SL CMT\n");
            l1++;
            continue;
    }
    if((line[l1][0]=='/') && (line[l1][1]=='*') && (flag == 0))
    {
            printf(" ML CMT STARTS\n");
            l1++;
            flag=1;
            continue;
    }
    if((line[l1][0]=='*') && (line[l1][1]=='/') && (flag == 1))
    {
            printf(" ML CMT ENDS\n");
            l1++;
            flag=0;
            continue;
    }
    if(flag)
    {
        l1++;
        continue;
    }
    token = strtok(line[l1]," ");
    if(strlen(token)==1 && token[0]=='\n')
        continue;
    while(token!=NULL)
    {
        if(check_keyword(token))
            printf(" KW ");
        else if(check_function(token))
            printf(" FC");
        else if(check_assign(token[0]))
            printf(" ASSIGN");
        else if(check_operator(token))
        {
            if(check_RELOP(token))
                printf(" RELOP");
            else if(check_LOGOP(token))
                printf(" LOGDOP");
            else if(check_ARTHOP(token))
                printf(" ARTHOP");
            else
```

```
                    printf(" OP");
            }
            else if(issymbol(token))
                printf(" SP");
            else if(check_num(token))
                printf(" NUMCONST");
            else if(check_char(token))
                printf(" CHARCONST");
            else if(!isdigit(token[0]))
                printf(" ID");
            else
                printf(" INVALID CHA");
            token = strtok(NULL," ");
        }
        printf("\n");
        l1++;
    }
}
void main()
{
    char code[100000];
    FILE * file = fopen("input.txt", "r");
    char  c;
    int idx=0;
    while (fscanf(file , "%c" ,&c) == 1)
    {
        code[idx] = c;
        idx++;
    }
    code[idx] = '\0';
    printf("\n");
    analyser(code);
}
```

(Sample input file)

```
/*
Multi line comment..
hi
hi
hi
*/
main()
{
int a = 5 , b = 10 ;
if ( a > b )
printf(\"a_is_greater\");
else
printf(\"b_is_greater\");
sum = add(a,b) ;
// This is a comment......
double 7aid ;
false = u > "8" ? a && b : a || b ;
}
```

(Output)

```
[Running] cd "g:\Academics\SSN\6th Sem\Compiler Design\Ex1\" && gcc sim
ple_analyser.c -
o simple_analyser && "g:\Academics\SSN\6th Sem\Compiler Design\Ex1\"sim
ple_analyser

 ML CMT STARTS
 ML CMT ENDS
 FC
 SP
 KW  ID ASSIGN NUMCONST SP ID ASSIGN NUMCONST SP
 KW  SP ID RELOP ID SP
 FC
 KW
 FC
 ID ASSIGN FC SP
 SL CMT
 KW  INVALID CHA SP
 ID ASSIGN ID RELOP CHARCONST RELOP ID LOGDOP ID SP ID LOGDOP ID SP
 SP

[Done] exited with code=18 in 2.238 seconds
```

Learning Outcome:

- I've learnt how the lexical analyser works and its basic functionalities.
- I've learnt how to tokenize an entire C program.
- I've learnt how to identify and group the lexemes into specific categories.
- 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.

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)

```
OUTPUT   DEBUG CONSOLE   TERMINAL

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.