A2-Implementation of Lexical Analyzer for the patterns using Lex

Sneha Sriram Kannan 185001157

19-02-2021

1 Problem Statement

Develop a Lexical analyzer to recognize the patterns namely, identifiers, constants, comments and operators using the following regular expressions using lex tool. Construct symbol table for the identifiers with the name, initial value, data type, size and location.

2 Code

```
/* To implement lexical analyzer using lex */
%{
    #include<stdio.h>
    #include<string.h>

    void addToSymbolTable(char id[]); //Function called whenever identifier is encountered
    void PrintSymbolTable(); //Displays the contents of the symbol table
    void addValueToTable(char value[]); //Value is found for the identifier so the value is added to the symbol table
    int notInSymbolTable(char id[]); //Checks if the identifier is already present in the symbol table
    void noValue(); //semicolon encountered so no value available for the identidier

    typedef struct SymbolTable{
        char id_name[50][20];
        int addr[50];
        int size[50];
        char type[50][10];
        char value[50][30];
        int row;
        int curraddr;
    }SymbolTable;

    char currType[10];
    int assignFlag=0;   //set when = is encountered
    int waitingForValue=0;  //set when identifier is encountered but value has not yet been defined

    SymbolTable *s;
%}

singleline [/]{2}.*
multiline "/*"([^*]*)"*/"
assign "="
keyword (if|else|do|while|void|int|float|char)
```

```
function [ ]*[a-zA-Z]+\(.*\).*
int [-+]?[0-9]+
float [-+]?[0-9]*[.][0-9]+
charconst \'.\'
strconst \".*\"
id [a-zA-z][a-zA-z0-9]*
arithop [+\-*/%]
relop [<>!=]{1,2}
logicalop &{2}|!|[|]{2}
seperator [();,{}]

%%
{multiline} {printf("MULTILINE ");}
{singleline} {printf("SINGLELINE ");}
{keyword} {printf("KW ");strcpy(currType,yytext);}
{function} {printf("FC ");}
{int} {printf("NUMCONST ");addValueToTable(yytext);}
{float} {printf("NUMCONST ");addValueToTable(yytext);}
{id} {printf("ID ");addToSymbolTable(yytext);}
{assign} {printf("ASSIGN ");assignFlag=1;}
{charconst} {printf("CHARCONST");addValueToTable(yytext);}
{strconst} {printf("STRCONST");addValueToTable(yytext);}
{arithop} {printf("ARITHOP ");}
{relop} {printf("RELOP ");}
{logicalop} {printf("LOGICALOP ");}
";" {printf("SP ");assignFlag=0;noValue();}
{seperator} {printf("SP ");}
"\n" {printf("\n");}
%%

int yywrap(void){}

int main()
{
    s=malloc(sizeof(SymbolTable));
    s->row=0;
    s->curraddr=3000;
    yyin = fopen("code.txt", "r");
    printf("\n==========================\nOUTPUT OF LEXICAL ANALYZER\n==========================\n");
    yylex();
    PrintSymbolTable();
    return 0;
}

void PrintSymbolTable(){
    printf("\n\n=======================\nContents of Symbol Table\n=======================\n");
    int i;
    printf("\n-----------------------------------------------\nName\tType\t\tValue\tSize\tAddress\n-----------------------------------------------\n");
    for(i=0;i<s->row;i++){
        printf("%s\t%s\t\t%s\t%d\t%d\n",s->id_name[i],s->type[i],s->value[i],s->size[i],s->addr[i]);
    }
}

void addToSymbolTable(char id[]){
    int index=s->row;
    if(notInSymbolTable(id)==1)
    {
        strcpy(s->id_name[index],id);
        s->addr[index]=s->curraddr;
        strcpy(s->type[index],currType);
        if(strcmp(s->type[index],"int")==0)
        {
            s->curraddr+=2;
            s->size[index]=2;
        }
        else if(strcmp(s->type[index],"char")==0)
        {
            s->curraddr+=1;
```

```
            s->size[index]=1;
        }
        else if(strcmp(s->type[index],"float")==0)
        {
            s->curraddr+=4;
            s->size[index]=4;
        }
        waitingForValue=1;
    }
}

int notInSymbolTable(char id[]){
    int i=0;
    //checks if the identifier has already been declared before
    for(i=0;i<s->row;i++){
        if(strcmp(id,s->id_name[i])==0){
            return 0;
        }
    }
    return 1;
}

void addValueToTable(char value[]){
    int index=s->row;
    //value for the identifier has been found so it is added to the symbol table
    if(assignFlag==1 && waitingForValue==1){
        assignFlag=0;
        waitingForValue=0;
        strcpy(s->value[index],value);
        s->row=s->row+1;
    }
}

void noValue(){
    int index=s->row;
    //semicolon encountered but no value assigned to the identifier
    if(waitingForValue==1)
    {
        strcpy(s->value[index],"NA");
        s->row=s->row+1;
    }
    waitingForValue=0;
}
```

3 Output Screenshot

A2 > code.txt

```
/* hello
sdf */
hello(){
  printf("Hello world");
}
main()
{
int a = 10 , b = 20;
char c = 'a' ;
float x = 2.34 ;
hello();
//check
int d;
if (a>b)
printf("a is greater");
else
printf("b is greater");
}
```

Figure 1: Input to Lexical Analyzer

```
============================
OUTPUT OF LEXICAL ANALYZER
============================
MULTILINE
FC
FC
SP
FC
SP
KW  ID  ASSIGN  NUMCONST  SP  ID  ASSIGN  NUMCONST SP
KW  ID  ASSIGN  CHARCONST SP
KW  ID  ASSIGN  NUMCONST  SP
FC
SINGLELINE
KW  ID SP
KW  SP ID RELOP ID SP
FC
KW
FC
SP

========================
Contents of Symbol Table
========================

-------------------------------------------------
Name    Type            Value   Size    Address
-------------------------------------------------
a       int             10      2       3000
b       int             20      2       3002
c       char            'a'     1       3004
x       float           2.34    4       3005
d       int             NA      2       3009
snehakannan@pop-os:~/Sneha/Semester 6/Compiler Design/Lab/A2$
```

Figure 2: Lexical Analyzer Output and Symbol Table

4 Learning Outcome

1. This assignment taught me how to use lex tool and how to run the programs
2. Since we first had to implement lexical analyzer using C and then lex tool, I understood how much easier it was to identify tokens using regular expressions in lex
3. I was able to parse the input to get the lexemes and identify the pattern that they matched.
4. The order in which regular expressions are specified is important.
5. I was able to generate the tokena and symbol table using lex tool