COMPILER DESIGN LAB 4

Date @January 27, 2026

Name: Devadathan N R
Registration No: 230905010
Class: CSE - A1
Roll No: 04

Question

Using getNextToken( ) implemented in Lab No 3, design a Lexical Analyser to implement the local

Symbol table looks like:

Lexeme name	Type	Size	returnType
FuncName	func	0	int/void/char
identifierName	int/char/float	4/8/0(func)	-

search() and insert() to prevent repeated entries.
hashing shld be implemented.

CODE:

la.h

This is an header file that declares the token structure and lexer function...it enables communication between the lexical analyzer and symbol table modules.

```
#ifndef LA_H
#define LA_H
```

```
#include <stdio.h>
#define MAX_LEXEME 100

struct token {
    char lexeme[MAX_LEXEME];
    char type[20];
    int row;
    int col;
};

struct token getNextToken(FILE *fp);

#endif
```

lexicalAnalyser.c:

```
#include "la.h"
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define BUFFER_SIZE 4096

static char buffer[BUFFER_SIZE];
static int bufSize = 0;
static int LB = 0, FP = 0;
static int row = 1, col = 1;
static int initialized = 0;

const char *keywords[] = {
 "int","float","double","char","void","long","short","signed","unsigned",
 "if","else","while","for","do","switch","case","default",
 "break","continue","return","goto",
```

```
 "auto","register","static","extern","const","volatile",
 "struct","union","enum","typedef","sizeof",
 "printf","scanf","malloc","free","strlen","strcpy","strcmp","exit",
 "main"
};

int keywordCount = sizeof(keywords)/sizeof(keywords[0]);


void loadBuffer(FILE *fp){
   bufSize = fread(buffer, 1, BUFFER_SIZE-1, fp);
   buffer[bufSize] = '\0';
}

char advanceChar(){
   char c = buffer[FP++];
   if(c=='\n'){ row++; col=1; }
   else col++;
   return c;
}

char peekChar(){ return buffer[FP]; }

int isKeyword(char *str){
   for(int i=0;i<keywordCount;i++)
      if(strcmp(str,keywords[i])==0) return 1;
   return 0;
}

int isSpecialSymbol(char c){
   return (c=='('||c==')'||c=='{'||c=='}'||c=='['||c==']'||c==';'||c==',');
}

void skipWhiteSpace(){
   while(peekChar()==' '||peekChar()=='\t'||peekChar()=='\n')
      advanceChar();
```

```
}

void skipSingleLineComment(){
   while(advanceChar()!='\n');
}

void skipMultiLineComment(){
   char c;
   while(FP<bufSize){
      c=advanceChar();
      if(c=='*' && peekChar()=='/'){
         advanceChar();
         break;
      }
   }
}

void skipPreprocessor(){
   while(advanceChar()!='\n');
}


struct token getNextToken(FILE *fp)
{
   struct token tk;
   if(!initialized){
      loadBuffer(fp);
      initialized=1;
   }

start:
   skipWhiteSpace();

   tk.row=row;
   tk.col=col;
```

```
    LB=FP;
    char c=advanceChar();

    if(c=='\0'){
        strcpy(tk.type,"EOF");
        return tk;
    }

    if(c=='#'){
        skipPreprocessor();
        goto start;
    }

    if(c=='/'){
        if(peekChar()=='/'){
            advanceChar();
            skipSingleLineComment();
            goto start;
        }
        if(peekChar()=='*'){
            advanceChar();
            skipMultiLineComment();
            goto start;
        }
    }

    /* string literal */
    if(c=='"'){
        while(peekChar()!='"' && peekChar()!='\0') advanceChar();
        advanceChar();

        int len=FP-LB;
        strncpy(tk.lexeme,buffer+LB,len);
        tk.lexeme[len]='\0';
        strcpy(tk.type,"STRING_LITERAL");
        return tk;
```

```
    }

    /* identifier or keyword */
    if(isalpha(c)||c=='_'){
        while(isalnum(peekChar())||peekChar()=='_')
            advanceChar();

        int len=FP-LB;
        strncpy(tk.lexeme,buffer+LB,len);
        tk.lexeme[len]='\0';

        if(isKeyword(tk.lexeme))
            strcpy(tk.type,"KEYWORD");
        else
            strcpy(tk.type,"ID");

        return tk;
    }

    /* number */
    if(isdigit(c)){
        while(isdigit(peekChar())) advanceChar();

        int len=FP-LB;
        strncpy(tk.lexeme,buffer+LB,len);
        tk.lexeme[len]='\0';

        strcpy(tk.type,"NUM");
        return tk;
    }

    /* special symbol */
    if(isSpecialSymbol(c)){
        tk.lexeme[0]=c;
        tk.lexeme[1]='\0';
        strcpy(tk.type,"SYMBOL");
```

```
        return tk;
    }

    /* unknown */
    tk.lexeme[0]=c;
    tk.lexeme[1]='\0';
    strcpy(tk.type,"UNKNOWN");

    return tk;
}
```

symbolTable.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "la.h"

#define TABLE_SIZE 101
#define MAX 100

struct symbol{
    char name[MAX];
    char type[20];
    int size;
    char returnType[20];
    struct symbol *next;
};

struct symbol *symTable[TABLE_SIZE]={NULL};

int hash(char *str){
    int h=0;
    while(*str)
```

```
        h=(h*31+*str++)%TABLE_SIZE;
    return h;
}

struct symbol* search(char *name){
    int index=hash(name);
    struct symbol *t=symTable[index];
    while(t){
        if(strcmp(t→name,name)==0) return t;
        t=t→next;
    }
    return NULL;
}

void insertSymbol(char *name,char *type,int size,char *ret){
    if(search(name)) return;

    int index=hash(name);
    struct symbol *n=malloc(sizeof(struct symbol));

    strcpy(n→name,name);
    strcpy(n→type,type);
    n→size=size;
    strcpy(n→returnType,ret);

    n→next=symTable[index];
    symTable[index]=n;
}

int getSize(char *d){
    if(strcmp(d,"int")==0) return 4;
    if(strcmp(d,"float")==0) return 4;
    if(strcmp(d,"double")==0) return 8;
    if(strcmp(d,"char")==0) return 1;
    return 0;
}
```

```
void printSymbolTable(){
   printf("\nLexeme        Type      Size     ReturnType\n");
   printf("-------------------------------------------\n");

   for(int i=0;i<TABLE_SIZE;i++){
      struct symbol *t=symTable[i];
      while(t){
         printf("%-15s %-10s %-8d %-10s\n",
               t->name,t->type,t->size,t->returnType);
         t=t->next;
      }
   }
}

int main(){
   FILE *fp=fopen("sample.c","r");
   if(!fp) return 0;

   struct token tk,nextTk;

   char currentType[20]="";
   int currentSize=0;

   while(1){
      tk=getNextToken(fp);
      if(strcmp(tk.type,"EOF")==0) break;

      if(strcmp(tk.type,"KEYWORD")==0){
         strcpy(currentType,tk.lexeme);
         currentSize=getSize(tk.lexeme);
      }
      else if(strcmp(tk.type,"ID")==0){
         nextTk=getNextToken(fp);

         if(strcmp(nextTk.type,"SYMBOL")==0 && strcmp(nextTk.lexeme,"(")==
```

```
0)
            insertSymbol(tk.lexeme,"func",0,currentType);
        else
            insertSymbol(tk.lexeme,currentType,currentSize,"-");
      }
   }

   fclose(fp);
   printSymbolTable();
   return 0;
}
```

sample.c

This is the sample input

```
#include <stdio.h>

/* global variables */
int a, b, c;
float price;
char grade;

int a;

/* function 1 */
int sum(int x, int y)
{
   int result;
   result = x + y;
   return result;
}

/* function 2 */
```

```
double average(double total, int count)
{
   double avg;
   avg = total / count;
   return avg;
}

/* function 3 */
void printMsg()
{
   char msg;
}

int inside;
```

OUTPUT:

```
CD_A1@CL3-02:~/Desktop/230905010_DevadathanNR_CS_A1/04_LAB$ gcc lexicalAnalyser.c symbolTable.c -o sym
CD_A1@CL3-02:~/Desktop/230905010_DevadathanNR_CS_A1/04_LAB$ ./sym

Lexeme          Type       Size     ReturnType
-----------------------------------------------
printMsg        func       0        void
avg             double     8        -
x               int        4        -
y               int        4        -
sum             func       0        int
count           int        4        -
result          int        4        -
average         func       0        double
price           float      4        -
msg             char       1        -
grade           char       1        -
total           double     8        -
inside          int        4        -
a               int        4        -
b               int        4        -
c               int        4        -
```

Text format:

```
Lexeme        Type      Size     ReturnType
-------------------------------------------
printMsg      func      0        void
avg           double    8        -
x             int       4        -
y             int       4        -
sum           func      0        int
count         int       4        -
result        int       4        -
average       func      0        double
price         float     4        -
msg           char      1        -
grade         char      1        -
total         double    8        -
inside        int       4        -
a             int       4        -
b             int       4        -
c             int       4        -
```