Name : Tushar Pathak
Registration Number : 230905396
Roll Number : CSE6B-48
Week Num : 4

LAB 4 : CONSTRUCTION OF SYMBOL TABLE

Q1) Using getNextToken( ) implemented in Lab No 3, design a Lexical Analyser to implement the following symbol tables.
a. local symbol table
b. global symbol table

Code:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

// TOKEN
typedef struct {
    char lexeme[30];
    char type[30];
    int row, col;
} Token;

// KEYWORDS
char *keywords[] = {
    "int", "char", "float", "if", "else", "while", "return", NULL
};

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

//LEXICAL ANALYZER
Token getNextToken(FILE *fp) {
    static int row = 1, col = 0;
    Token t;
    char c;
    int i;

    while ((c = fgetc(fp)) != EOF) {
        col++;

        if (c == '\n') {
            row++;
```

```
    col = 0;
    continue;
}

// Ignore preprocessor directives
if (c == '#') {
    while ((c = fgetc(fp)) != '\n' && c != EOF);
    row++;
    col = 0;
    continue;
}

//Ignore comments
if (c == '/') {
    char next = fgetc(fp);
    if (next == '/') {              // single-line comment
        while ((c = fgetc(fp)) != '\n' && c != EOF);
        row++;
        col = 0;
        continue;
    } else if (next == '*') {       // multi-line comment
        char prev = 0;
        while ((c = fgetc(fp)) != EOF) {
            if (prev == '*' && c == '/')
                break;
            prev = c;
        }
        continue;
    } else {
        ungetc(next, fp);
    }
}

//Ignore string literals
if (c == '"') {
    while ((c = fgetc(fp)) != '"' && c != EOF);
    continue;
}

// Identifier or Keyword
if (isalpha(c)) {
    i = 0;
    t.lexeme[i++] = c;
    while (isalnum(c = fgetc(fp))) {
        t.lexeme[i++] = c;
        col++;
    }
    t.lexeme[i] = '\0';
    ungetc(c, fp);

    strcpy(t.type, isKeyword(t.lexeme) ? "KEYWORD" : "IDENTIFIER");
    t.row = row;
```

```
            t.col = col - strlen(t.lexeme) + 1;
            return t;
        }

        // Number
        if (isdigit(c)) {
            i = 0;
            t.lexeme[i++] = c;
            while (isdigit(c = fgetc(fp))) {
                t.lexeme[i++] = c;
                col++;
            }
            t.lexeme[i] = '\0';
            ungetc(c, fp);

            strcpy(t.type, "NUMBER");
            t.row = row;
            t.col = col - strlen(t.lexeme) + 1;
            return t;
        }

        // Operators
        if (strchr("+-*/<>=!", c)) {
            t.lexeme[0] = c;
            t.lexeme[1] = '\0';
            strcpy(t.type, "OPERATOR");
            t.row = row;
            t.col = col;
            return t;
        }

        // Special Symbols
        if (strchr(";(){}[],", c)) {
            t.lexeme[0] = c;
            t.lexeme[1] = '\0';
            strcpy(t.type, "SPECIAL SYMBOL");
            t.row = row;
            t.col = col;
            return t;
        }
    }

    strcpy(t.type, "EOF");
    return t;
}

//SYMBOL TABLES
#define MAX 100

typedef struct {
    char lexemeName[30];
    char tokenType[20];
```

```
    int ptrToSymTabEntry;
} GlobalSymTab;

typedef struct {
    char lexemeName[30];
    char type[20];
    int size;
} LocalSymTab;

GlobalSymTab GST[MAX];
LocalSymTab LST[MAX];

int gstIndex = 0;
int lstIndex = 0;

// SYMBOL TABLE FUNCTIONS
int getSize(char *type) {
    if (strcmp(type, "int") == 0) return 4;
    if (strcmp(type, "char") == 0) return 1;
    if (strcmp(type, "float") == 0) return 4;
    return 0;
}

void insertGlobal(char *lexeme, char *tokenType) {
    for (int i = 0; i < gstIndex; i++) {
        if (strcmp(GST[i].lexemeName, lexeme) == 0)
            return;
    }
    strcpy(GST[gstIndex].lexemeName, lexeme);
    strcpy(GST[gstIndex].tokenType, tokenType);
    GST[gstIndex].ptrToSymTabEntry = -1;
    gstIndex++;
}

void insertLocal(char *lexeme, char *type) {
    strcpy(LST[lstIndex].lexemeName, lexeme);
    strcpy(LST[lstIndex].type, type);
    LST[lstIndex].size = getSize(type);
    lstIndex++;
}


int main() {
    FILE *fp = fopen("Q1Input.c", "r");
    Token t;
    int scope = 0;        // 0 = global, 1 = local
    char currentType[20] = "";

    if (!fp) {
        printf("File not found!\n");
        return 0;
    }
```

```
    while (strcmp((t = getNextToken(fp)).type, "EOF") != 0) {

        if (strcmp(t.lexeme, "{") == 0)
            scope = 1;

        if (strcmp(t.lexeme, "}") == 0)
            scope = 0;

        if (strcmp(t.type, "KEYWORD") == 0 &&
           (strcmp(t.lexeme, "int") == 0 ||
            strcmp(t.lexeme, "char") == 0 ||
            strcmp(t.lexeme, "float") == 0)) {
            strcpy(currentType, t.lexeme);
        }

        if (strcmp(t.type, "IDENTIFIER") == 0 && strcmp(currentType, "") != 0) {
            if (scope == 0)
                insertGlobal(t.lexeme, t.type);
            else
                insertLocal(t.lexeme, currentType);
        }

        if (strcmp(t.lexeme, ";") == 0)
            strcpy(currentType, "");
    }

    fclose(fp);


    printf("\nGLOBAL SYMBOL TABLE\n");
    printf("Lexeme\tTokenType\tPtr\n");
    for (int i = 0; i < gstIndex; i++) {
        printf("%s\t%s\t\t%d\n",
               GST[i].lexemeName,
               GST[i].tokenType,
               GST[i].ptrToSymTabEntry);
    }

    printf("\nLOCAL SYMBOL TABLE\n");
    printf("Lexeme\tType\tSize\n");
    for (int i = 0; i < lstIndex; i++) {
        printf("%s\t%s\t%d\n",
               LST[i].lexemeName,
               LST[i].type,
               LST[i].size);
    }

    return 0;
}
```

Input File:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
   int a, b;
   printf("Enter the numbers A and B...");
   scanf("%d",&a,&b);
   if(a>b){
      printf("%d",a);
      return a;
   }else{
      printf("%d",b);
      return b;
   }
}
```

Output:

```
cdl-6cse-b2@sce-cl11-15:~/Desktop/230905396/Lab4$ gcc Q1.c -o Q1
cdl-6cse-b2@sce-cl11-15:~/Desktop/230905396/Lab4$ ./Q1

GLOBAL SYMBOL TABLE
Lexeme  TokenType       Ptr
main    IDENTIFIER              -1

LOCAL SYMBOL TABLE
Lexeme  Type    Size
a       int     4
b       int     4
```