Saivya Singh
CSE D 44
220905370

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

Code:

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

#define MAX_SYMBOL_TABLE_SIZE 100

struct token {
int row;
int col;
char token_name[100];
char type[50];
int size;
};

struct symbol {
char name[100];
char type[50];
int size;
int identifierNumber;
struct symbol *next;
};

struct symbol *symbolTable[MAX_SYMBOL_TABLE_SIZE];

int hash(const char *str) {
int hashValue = 0;
while (*str) {
hashValue = (hashValue * 31 + *str++) % MAX_SYMBOL_TABLE_SIZE;
}
return hashValue;
}

void insertSymbol(const char *name, const char *type, int size, int identifierNumber) {
int index = hash(name);
struct symbol *newSymbol = malloc(sizeof(struct symbol));
strcpy(newSymbol->name, name);
strcpy(newSymbol->type, type);
```

```
newSymbol->size = size;
newSymbol->identifierNumber = identifierNumber;
newSymbol->next = symbolTable[index];
symbolTable[index] = newSymbol;
}

struct symbol *lookupSymbol(const char *name) {
int index = hash(name);
struct symbol *current = symbolTable[index];
while (current) {
if (strcmp(current->name, name) == 0) {
return current;
}
current = current->next;
}
return NULL;
}

int isKeyword(const char *word) {
const char *keywords[] = {"int", "float", "if", "else", "while", "return", "bool", "void"};
for (int i = 0; i < 8; i++) {
if (strcmp(word, keywords[i]) == 0) {
return 1;
}
}
return 0;
}

void fillToken(struct token *tkn, int row, int col, const char *name, int size) {
tkn->row = row;
tkn->col = col;
strcpy(tkn->token_name, name);
tkn->size = size;
}

int getDataTypeSize(const char *type) {
if (strcmp(type, "int") == 0) {
return sizeof(int);
} else if (strcmp(type, "bool") == 0) {
return sizeof(char); // bool is typically a single byte
}
return 0;
}

struct token getNextToken(FILE *fin, int *row, int *col) {
int c, d;
struct token tkn = {.row = -1};
int gotToken = 0;

while (!gotToken && (c = getc(fin)) != EOF) {
if (c == '\n') {
```

```
++(*row);
*col = 0;
continue;
}

if (isalpha(c) || c == '_') {
tkn.row = *row;
tkn.col = *col;
tkn.token_name[0] = c;
int k = 1;
while ((c = getc(fin)) != EOF && (isalnum(c) || c == '_')) {
tkn.token_name[k++] = c;
++(*col);
}
tkn.token_name[k] = '\0';
strcpy(tkn.type, isKeyword(tkn.token_name) ? "Keyword" : "Identifier");
tkn.size = k;

gotToken = 1;
fseek(fin, -1, SEEK_CUR);
}
else if (isdigit(c)) {
tkn.row = *row;
tkn.col = *col;
tkn.token_name[0] = c;
int k = 1;
while ((c = getc(fin)) != EOF && isdigit(c)) {
tkn.token_name[k++] = c;
++(*col);
}
tkn.token_name[k] = '\0';
strcpy(tkn.type, "Number");
tkn.size = k;

gotToken = 1;
fseek(fin, -1, SEEK_CUR);
}
else {
++(*col);
}
}
return tkn;
}

void printSymbolTable(FILE *fout) {
fprintf(fout, "Lexeme Name Type Size Identifier Number\n");
fprintf(fout, "-------------------------------------------------\n");
for (int i = 0; i < MAX_SYMBOL_TABLE_SIZE; i++) {
struct symbol *current = symbolTable[i];
while (current) {
```

```
fprintf(fout, "%-15s %-7s %-5d %-20d\n", current->name, current->type, current->size,
current->identifierNumber);
current = current->next;
}
}
}

int main() {
int row = 1, col = 0;
for (int i = 0; i < MAX_SYMBOL_TABLE_SIZE; i++) {
symbolTable[i] = NULL;
}

FILE *fin = fopen("input.c", "r");
if (fin == NULL) {
printf("File not found!\n");
return 1;
}

FILE *fout = fopen("output.txt", "w");
if (fout == NULL) {
printf("Unable to open output file.\n");
fclose(fin);
return 1;
}

int identifierNumber = 1;
while (1) {
struct token t = getNextToken(fin, &row, &col);
if (t.row == -1) break;

if (strcmp(t.type, "Identifier") == 0) {
struct symbol *existingSymbol = lookupSymbol(t.token_name);
if (!existingSymbol) {
insertSymbol(t.token_name, t.type, t.size, identifierNumber);
identifierNumber++;
}
}
}

printSymbolTable(fout);
fclose(fin);
fclose(fout);
return 0;
}
```

Input:

```
int sum(int a, int b)
{ int s=a+b;
return s;
}
bool search(int *arr,int key)
{
int i;
for(i=0;i<10;i++){
if(arr[i]==key)
return true;
else return false;
}
}
void main()
{
int a[20],i,sum;
bool status;
printf("Enter array elements:");
for(i=0;i<10;++i)
scanf("%d",&a[i]);
sum=a[0]+a[4];
status=search(a,sum);
printf("%d",status);
}
```

Output:

Lexeme Name	Type	Size	Identifier Number
sum	Func	3	1
a	Int	1	2
b	Int	1	3
s	Identifier	1	4
search	Func	6	5
arr	Identifier	3	6
key	Identifier	3	7
i	Int	1	8
for	Identifier	3	9
true	Identifier	4	10
false	Identifier	5	11
main	Func	4	12
a	Int	4	80
status	Identifier	6	13
printf	Identifier	6	14

Enter	Identifier	5	15
array	Identifier	5	16
elements	Identifier	8	17
scanf	Identifier	5	18