Ayush Goyal
190905522 CSE D Roll 62

Compiler Design Lab 4: Construction of Symbol Table

1. Using getNextToken( ) implemented in Lab No 3,design a Lexical Analyser to implement local and global symbol table to store tokens for identifiers using array of structure.

The below code has been modified from the one done in lab 3 to add symbol table functionality as well.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#define MAX_SIZE 20

int removeSpaces(){
	FILE *fa, *fb;
	int ca, cb;
	fa = fopen("input.c", "r");
	if(fa == NULL){
		printf("Cannot open file!\n");
		exit(0);
	}
	fb = fopen("output1.c", "w");
	ca = getc(fa);
	while(ca != EOF){
		if(ca == ' ' || ca == '\t'){
			putc(' ',fb);
			while(ca == ' ' || ca == '\t')
				ca = getc(fa);
		}
		if(ca == '/'){
			cb = getc(fa);
			if(cb == '/'){
				while(ca != '\n')
					ca = getc(fa);
			}
			else if(cb == '*'){
				do{
					while(ca != '*')
						ca = getc(fa);
					ca = getc(fa);
				}while(ca != '/');
			}
			else{
				putc(ca, fb);
```

```
                    putc(cb, fb);
                }
            }
            else
                putc(ca, fb);
            ca = getc(fa);
        }
        fclose(fa);
        fclose(fb);
        return 0;
}

int preProcessIgnore(){
        FILE *fin = fopen("output1.c", "r");
        char c = 0;
        char buffer[100];
        buffer[0] = '\0';
        int i = 0;
        char *includeStr = "include", *defineStr = "define", *mainStr = "main";
        int mainFlag = 0, rowNum = 1;
        while(c != EOF){
            c = fgetc(fin);
            if(c == '#' && mainFlag == 0){
                c = 'a';
                while(isalpha(c) != 0){
                    c = fgetc(fin);
                    buffer[i++] = c;
                }
                buffer[i] = '\0';
                if(strstr(buffer, includeStr) != NULL || strstr(buffer, defineStr) != NULL){
                    rowNum++;
                    while(c!='\n'){
                        c = fgetc(fin);
                    }
                }
                else{
                    for(int j=0;j<i;j++);
                    while(c!='\n'){
                        c = fgetc(fin);
                    }
                }
                i = 0;
                buffer[0]= '\0';
            }
            else{
                if(mainFlag == 0){
                    buffer[i++] = c;
                    buffer[i] = '\0';
                    if(strstr(buffer, mainStr) != NULL){
                        mainFlag = 1;
                    }
                }
```

```
                        if(c == ' ' || c == '\n'){
                                buffer[0] = '\0';
                                i = 0;
                        }
                }
        }
        fclose(fin);
        return rowNum;
}

char keywords[32][10] = {"auto", "double", "int", "struct",
"break", "else", "long", "switch", "case", "enum", "register",
"typedef", "char", "extern", "return", "union", "const", "float",
"short", "unsigned", "continue", "for", "signed", "void",
"default", "goto", "sizeof", "voltile", "do", "if", "static",
"while"}; // list of keywords

char operators[5] = {'+','-','/','%','*'};
char brackets[6] = {'(', ')', '[', ']', '{', '}'};
char data_types[][10] = {"double", "int", "char", "float"};
char special_symbols[12] = {'*', ';', ':', '.', ',', '^', '&',
'!', '>', '<', '~', '`'};

//lexeme type enumerator
enum TYPE{
        IDENTIFIER,
        KEYWORD,
        STRING_LITERAL,
        NUMERIC_CONSTANT,
        OPERATOR,
        BRACKET,
        SPECIAL_SYMBOL,
        RELATIONAL_OPERATOR,
        CHARACTER_CONSTANT
};

//map for the type to string
char types[][30] = {"IDENTIFIER", "KEYWORD", "STRING_LITERAL",
"NUMERIC_CONSTANT", "OPERATOR", "BRACKET", "SPECIAL_SYMBOL",
"RELATIONAL_OPERATOR", "CHARACTER_CONSTANT"};


//element structure for hash table created:

typedef struct node{
        char *cur;
        int row,col;
        struct node *next;
        enum TYPE type;
} *Node;

//element structure for symbol table created:
```

```
typedef struct symbol{
	char *name;
	char *data_type;
	struct symbol *next;
	unsigned int size;
} *Symbol;

Node hashTable[MAX_SIZE]; //hash table created
Symbol st[MAX_SIZE]; //symbol table created

//to check for comparision

int isKeyword(char buffer[]){
	for(int i=0;i<32;i++){
		if(strcmp(buffer, keywords[i]) == 0){
			return 1;
		}
	}
	return 0;
}

int isDatatype(char buffer[]){
	for(int i=0;i<4;i++){
		if(strcmp(buffer, data_types[i]) == 0){
			return 1;
		}
	}
	return 0;
}

int isOperator(char ch){
	for(int i=0;i<5;i++){
		if(operators[i] == ch)
			return 1;
	}
	return 0;
}

int isBracket(char ch){
	for(int i=0;i<6;i++){
		if(brackets[i] == ch)
			return 1;
	}
	return 0;
}

int isSpecialSymbol(char c){
	for(int i=0;i<12;i++){
		if(special_symbols[i] == c)
			return 1;
	}
	return 0;
}

//creating a hashing function
int hash(int size){
	return (size % MAX_SIZE);
}

void displaySymbolTable(){
	printf("\nSymbol table (in the format < Name , Type , Size > ): \n\n");
	for(int i=0;i<MAX_SIZE;i++){
		if(st[i] == NULL)
			continue;
		else{
			Symbol cur = st[i];
			while(cur){
				printf(" %8s\t %8s\t %8d\t \n",cur->name, cur->data_type, cur->size);
				cur = cur->next;
			}
		}
	}
}

//searching in the symbol table
int searchSymbolTable(char identifier[], char data_type[]){
	int index = hash(strlen(identifier));
	if(st[index] == NULL)
		return -1;
	Symbol cur = st[index];
	int i = 0;
	while(cur != NULL){
		if(strcmp(identifier, cur->name) == 0)
			return i;
		cur = cur->next;
		i++;
	}
	return -1;
}

//searching in the hash table
int searchHashTable(char buffer[], enum TYPE type){
	int index = hash(strlen(buffer));
	if(hashTable[index] == NULL)
		return 0;
	Node cur = hashTable[index];
	while(cur != NULL){
		if(strcmp(cur->cur, buffer) == 0)
			return 1;
		cur = cur->next;
	}
	return 0;
}
```

```
//inserting in the symbol table
void insertSymbolTable(char identifier[], char data_type[]){
	if(searchSymbolTable(identifier, data_type) == -1){
		Symbol n = (Symbol)malloc(sizeof(struct symbol));
		char *str = (char *)calloc(strlen(identifier)+1, sizeof(char));
		strcpy(str, identifier);
		n->name = str;
		n->next = NULL;
		char *typee = (char *)calloc(strlen(data_type)+1, sizeof(char));
		strcpy(typee, data_type);
		n->data_type = typee;
		if(strcmp(data_type, "int") == 0)
			n->size = 4;
		else if(strcmp(data_type, "double") == 0)
			n->size = 8;
		else if(strcmp(data_type, "char") == 0)
			n->size = 1;
		else if(strcmp(data_type, "function") == 0)
			n->size = 0;
		else
			n->size = 4;
		int index = hash(strlen(identifier));
		if(st[index] == NULL){
			st[index] = n;
			return;
		}
		Symbol cur = st[index];
		while(cur->next != NULL)
			cur = cur->next;
		cur->next = n;
	}
}

//inserting elements in the hash table
void insertHashTable(char buffer[], int row, int col, enum TYPE type){
	if(type == IDENTIFIER || searchHashTable(buffer, type) == 0){
			printf("<%s,%d,%d,%s>\n", buffer, row, col,types[type]);
			int index = hash(strlen(buffer));
			Node n = (Node)malloc(sizeof(struct node));
			char *str = (char *)calloc(strlen(buffer) + 1, sizeof(char));
			strcpy(str, buffer);
			n->cur = str;
			n->next = NULL;
			n->row = row;
			n->col = col;
			n->type = type;
			if(hashTable[index] == NULL){
				hashTable[index] = n;
				return;
			}
			Node cur = hashTable[index];
```

```
				while(cur->next != NULL){
					cur = cur->next;
				}
				cur->next = n;
		}
}


int main(){
		//removing all extra tabs and whitespaces
		removeSpaces();
		//getting the row(line) after ignoring all preprocessive directives
		int rowNum = preProcessIgnore();

		enum TYPE type;

		for(int i=0;i<MAX_SIZE;i++)
			hashTable[i] = NULL;

		FILE *fin = fopen("output1.c", "r");
		if(fin == NULL){
			printf("Cannot open file!\n");
			return 0;
		}

		char buffer[100], c = 0, data_type_buffer[100];
		int i = 0, col_global = 1, col, temp_row = --rowNum;
		while(temp_row > 0){
			c = fgetc(fin);
			if(c == '\n')
				temp_row--;
		}

		while(c != EOF){
			if(isalpha(c) != 0 || c == '_'){
				buffer[i++] = c;
				col = col_global;
				while(isalpha(c) != 0 || c == '_' || isdigit(c) != 0){
					c = fgetc(fin);
					col_global++;
					if (isalpha(c) != 0 || c == '_' || isdigit(c) != 0)
						buffer[i++] = c;
				}

				buffer[i] = '\0';

				if(isDatatype(buffer) == 1){
					insertHashTable(buffer, rowNum, col-1, KEYWORD);
					strcpy(data_type_buffer, buffer);
				}
				else if(isKeyword(buffer) == 1){
					insertHashTable(buffer, rowNum, col-1, KEYWORD);
			}
			else{
				insertHashTable(buffer, rowNum, col-1, IDENTIFIER);
				if(c == '(')
					insertSymbolTable(buffer, "function");
				else
					insertSymbolTable(buffer, data_type_buffer);
				data_type_buffer[0] = '\0';
			}

			i = 0;
			if(c == '\n')
				rowNum++, col_global = 1;
			buffer[0] = '\0';
		}
		else if(isdigit(c) != 0){
			buffer[i++] = c;
			col = col_global;
			while(isdigit(c) != 0 || c == '.'){
				c = fgetc(fin);
				col_global++;
				if(isdigit(c) != 0 || c == '.')
					buffer[i++] = c;
			}
			buffer[i] = '\0';
			insertHashTable(buffer, rowNum, col-1, NUMERIC_CONSTANT); //
numerical constant
			i = 0;
			if(c == '\n')
				rowNum++, col_global = 1;
			buffer[0] = '\0';
		}
		else if(c == '\"'){
			col = col_global;
			buffer[i++] = c;
			c = 0;
			while(c != '\"'){
				c = fgetc(fin);
				col_global++;
				buffer[i++] = c;
			}
			buffer[i] = '\0';
			insertHashTable(buffer, rowNum, col-1, STRING_LITERAL); // string
literals
			buffer[0] = '\0';
			i = 0;
			c = fgetc(fin);
			col_global++;
		}
		else if(c == '\''){
			col = col_global;
			buffer[i++] = c;
```

```
                        c = 0;
                        c = fgetc(fin);
                        col_global++;
                        buffer[i++] = c;
                        if(c == '\\'){
                                c = fgetc(fin);
                                col_global++;
                                buffer[i++] = c;
                        }
                        c = fgetc(fin);
                        col_global++;
                        buffer[i++] = c;
                        buffer[i] = '\0';
                        insertHashTable(buffer, rowNum, col-1, CHARACTER_CONSTANT);
//character constants
                        buffer[0] = '\0';
                        i=0;
                        c = fgetc(fin);
                        col_global++;
                }
                else{
                        col = col_global;
                        if(c == '='){ // relational and logical operators
                                c = fgetc(fin);
                                col_global++;
                                if(c == '='){
                                        insertHashTable("==", rowNum, col-1,
RELATIONAL_OPERATOR);
                                }
                                else{
                                        insertHashTable("=", rowNum, col-1,
RELATIONAL_OPERATOR);
                                        fseek(fin, -1, SEEK_CUR);
                                        col_global--;
                                }
                        }
                        else if(c == '>' || c == '<' || c == '!'){
                                char temp = c;
                                c = fgetc(fin);
                                col_global++;
                                if(c == '='){
                                        char temp_str[3] = {temp, '=', '\0'};
                                        insertHashTable(temp_str, rowNum, col-1,
RELATIONAL_OPERATOR);
                                }
                                else{
                                        char temp_str[2] = {temp, '\0'};
                                        insertHashTable(temp_str, rowNum, col-1,
RELATIONAL_OPERATOR);
                                        fseek(fin, -1, SEEK_CUR);
                                        col_global--;
                                }
```

```
				}
				else if(isBracket(c) == 1){
					char temp_string[2] = {c,'\0'};
					insertHashTable(temp_string, rowNum, col-1, BRACKET);
				}
				else if(isSpecialSymbol(c) == 1){
					char temp_string[2] = {c,'\0'};
					insertHashTable(temp_string, rowNum, col-1, SPECIAL_SYMBOL);
				}
				else if(isOperator(c) == 1){
					char temp = c;
					c = fgetc(fin);
					col_global++;
					if (c == '=' || (temp == '+' && c == '+') || (temp == '-' && c == '-')){
						char temp_string[3] = {temp,c,'\0'};
						insertHashTable(temp_string, rowNum, col-1, OPERATOR);
					}
					else{
						char temp_String[2] = {temp,'\0'};
						insertHashTable(temp_String, rowNum, col-1, OPERATOR);
						fseek(fin, -1, SEEK_CUR);
						col_global--;
					}
				}
				else if(c == '\n')
					rowNum++, col_global = 1;
				c = fgetc(fin);
				col_global++;
			}
	}
	displaySymbolTable();
	return 0;
}
```

I have made a customized “input.c” file to test the program output which is executed and shown below. The “input.c” file is shown below but the main output of the code is shown in the output screenshot:

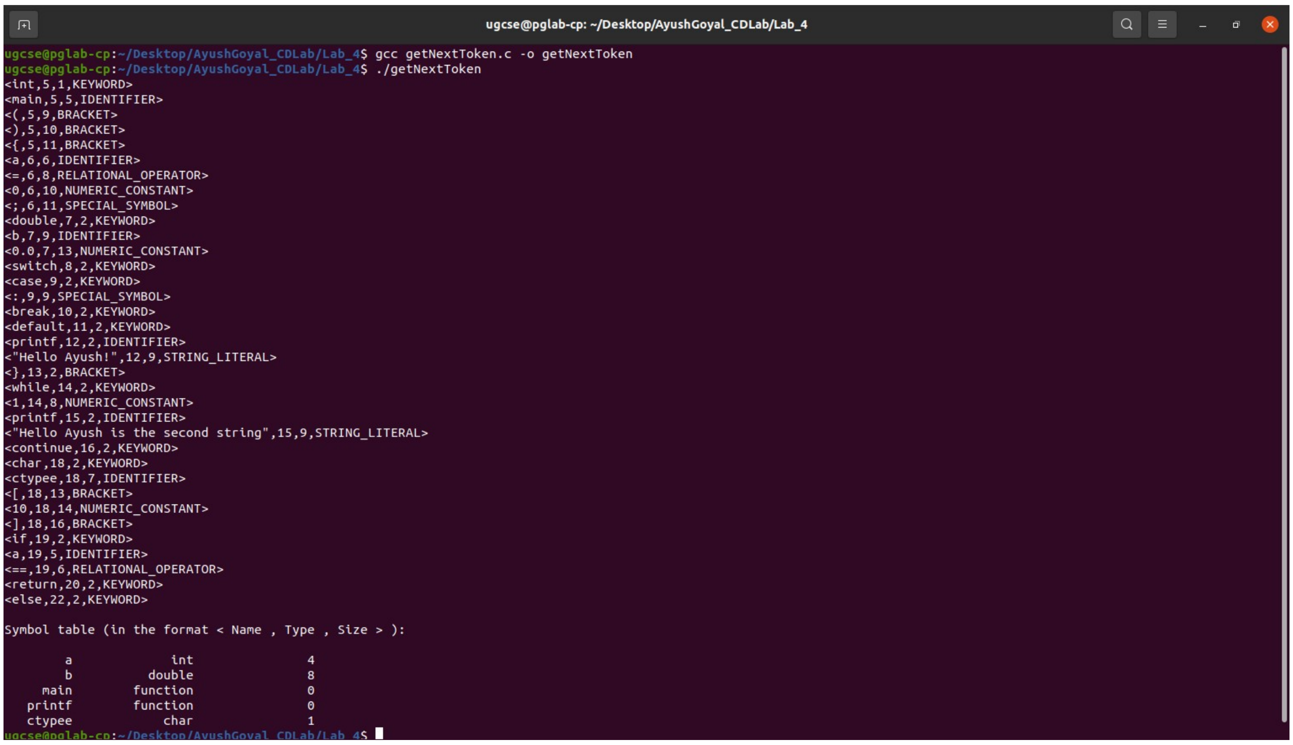
“input.c”:

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

int main(){
	int a = 0;
	double b = 0.0;
	switch(0){
		case 0 :
					break;
		default :
					printf("Hello Ayush!");
```

```
        }
        while(1){
                printf("Hello Ayush is the second string");
                continue;
        }
        char ctypee[10];
        if(a==1){
                return 0;
        }
        else
                return 1;
        return 0;
}
```

Output:



```
ugcse@pglab-cp:~/Desktop/AyushGoyal_CDLab/Lab_4$ gcc getNextToken.c -o getNextToken
ugcse@pglab-cp:~/Desktop/AyushGoyal_CDLab/Lab_4$ ./getNextToken
<int,5,1,KEYWORD>
<main,5,5,IDENTIFIER>
<(,5,9,BRACKET>
<),5,10,BRACKET>
<{,5,11,BRACKET>
<a,6,6,IDENTIFIER>
<=,6,8,RELATIONAL_OPERATOR>
<0,6,10,NUMERIC_CONSTANT>
<;,6,11,SPECIAL_SYMBOL>
<double,7,2,KEYWORD>
<b,7,9,IDENTIFIER>
<0.0,7,13,NUMERIC_CONSTANT>
<switch,8,2,KEYWORD>
<case,9,2,KEYWORD>
<:,9,9,SPECIAL_SYMBOL>
<break,10,2,KEYWORD>
<default,11,2,KEYWORD>
<printf,12,2,IDENTIFIER>
<"Hello Ayush!",12,9,STRING_LITERAL>
<},13,2,BRACKET>
<while,14,2,KEYWORD>
<1,14,8,NUMERIC_CONSTANT>
<printf,15,2,IDENTIFIER>
<"Hello Ayush is the second string",15,9,STRING_LITERAL>
<continue,16,2,KEYWORD>
<char,18,2,KEYWORD>
<ctypee,18,7,IDENTIFIER>
<[,18,13,BRACKET>
<10,18,14,NUMERIC_CONSTANT>
<],18,16,BRACKET>
<if,19,2,KEYWORD>
<a,19,5,IDENTIFIER>
<==,19,6,RELATIONAL_OPERATOR>
<return,20,2,KEYWORD>
<else,22,2,KEYWORD>

Symbol table (in the format < Name , Type , Size > ):

        a            int                4
        b         double                8
     main       function                0
   printf       function                0
   ctypee           char                1
ugcse@pglab-cp:~/Desktop/AyushGoyal_CDLab/Lab_4$
```

THE END