# Lab 4 – Construction of Symbol Table

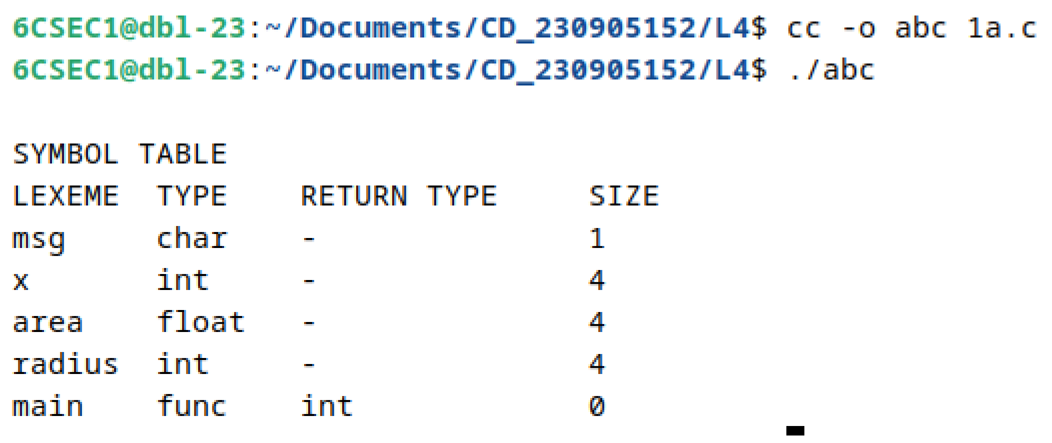
Q.

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define TableLength 50

FILE *fp;
int row = 1, col = 0;
char *keywords[] = {"int", "if", "else", "while", "return", “for”};
int kw_count = 6;
char lookahead = '\0';
int has_lookahead = 0;

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

int isArithmetic(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/');
}
int isRelop(char c) {
    return (c == '<' || c == '>' || c == '=' || c == '!');
}
int isLogic(char c) {
    return (c == '&' || c == '|');
}
int isSpecial(char c) {
    return (c == '(' || c == ')' || c == '{' || c == '}' || c == ';' || c ==
',');
}

char getChar() {
    if (has_lookahead) {
        has_lookahead = 0;
        return lookahead;
    }
    return fgetc(fp);
}

void putBack(char c) {
    lookahead = c;
    has_lookahead = 1;
}

void skipPreprocessor() {
    char w[16], c;
    int i = 0;
    while ((c = getChar()) == ' ');
    while (isalpha(c)) {
        w[i++] = c;
        c = getChar();
    }
    w[i] = '\0';
    if (!strcmp(w, "include") || !strcmp(w, "define")) {
        while (c != '\n' && c != EOF)
            c = getChar();
        row++;
        col = 0;
    } else {
        putBack(c);
    }
}

void skipComment(char first) {
    char c;
    if (first == '/') {
        while ((c = getChar()) != '\n' && c != EOF);
        row++;
        col = 0;
    } else {
        while (1) {
            c = getChar();
            if (c == '\n') {
                row++;
                col = 0;
            }
            if (c == '*' && getChar() == '/')
                break;
        }
    }
}

void skipString() {
    char c;
    while ((c = getChar()) != '"' && c != EOF);
}

struct symbol {
    char lexeme[50];
    char type[30];
    char returnType[30];
    int size;
};

struct ListElement {
    struct symbol sym;
    struct ListElement *next;
};

struct ListElement *TABLE[TableLength];

void Initialize() {
    for (int i = 0; i < TableLength; i++)
        TABLE[i] = NULL;
}

int HASH(char *str) {
    int hash = 0;
    while (*str) {
        hash = hash * 26 + *str;
        str++;
    }
    return hash % TableLength;
}

int SEARCH(char *lex) {
    int index = HASH(lex);
    struct ListElement *temp = TABLE[index];
    while (temp != NULL) {
        if (!strcmp(temp->sym.lexeme, lex))
            return 1;
        temp = temp->next;
    }
    return 0;
}

int getSize(char *type) {
    if (!strcmp(type, "int")) return 4;
    if (!strcmp(type, "float")) return 4;
    if (!strcmp(type, "char")) return 1;
    if (!strcmp(type, "bool")) return 1;
    if (!strcmp(type, "void")) return 0;
    if (strstr(type, "*")) return 8;
    if (strstr(type, "[]")) return 8;
    return 0;
}

void INSERT(char *lex, char *type, char *returnType) {
    if (SEARCH(lex)) return;
    if (strlen(type) == 0) return;
    struct symbol s;
    strcpy(s.lexeme, lex);
    strcpy(s.type, type);
    strcpy(s.returnType, returnType);
    s.size = getSize(type);
    int index = HASH(lex);
    struct ListElement *cur = malloc(sizeof(struct ListElement));
    cur->sym = s;
    cur->next = NULL;
    if (TABLE[index] == NULL)
        TABLE[index] = cur;
    else {
        struct ListElement *temp = TABLE[index];
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = cur;
    }
}

void Display() {
    printf("\nSYMBOL TABLE\n");
    printf("LEXEME\tTYPE\tRETURN TYPE\tSIZE\n");
    for (int i = 0; i < TableLength; i++) {
        struct ListElement *temp = TABLE[i];
        while (temp != NULL) {
            printf("%s\t%s\t%s\t\t%d\n",
                   temp->sym.lexeme,
                   temp->sym.type,
                   temp->sym.returnType,
                   temp->sym.size);
            temp = temp->next;
        }
    }
}

int getNextToken() {
    char c, buf[100];
    int i;
    static char currentType[50] = "";
    while ((c = getChar()) != EOF) {
        col++;
        if (c == '\n') {
            row++;
            col = 0;
            continue;
        }
        if (c == '#') {
            skipPreprocessor();
            continue;
        }
        if (c == '/') {
            char next = getChar();
            if (next == '/' || next == '*') {
                skipComment(next);
                continue;
            }
            putBack(next);
        }
        if (c == '"') {
            skipString();
            continue;
        }
        if (isalpha(c)) {
            i = 0;
            buf[i++] = c;
            while (isalnum(c = getChar()) || c == '_') {
                buf[i++] = c;
                col++;
            }
            buf[i] = '\0';
            putBack(c);
            if (!strcmp(buf, "int") || !strcmp(buf, "float") ||
                !strcmp(buf, "char") || !strcmp(buf, "bool") ||
                !strcmp(buf, "void")) {
                strcpy(currentType, buf);
            } else if (!isKeyword(buf)) {
                char next = getChar();
                putBack(next);
                if (next == '(') {
                    if (strlen(currentType) > 0 && strcmp(buf, "printf") != 0 &&
strcmp(buf, "scanf") != 0)
                        INSERT(buf, "func", currentType);
                } else {
                    if (strlen(currentType) > 0)
                        INSERT(buf, currentType, "-");
                }
            }
            return 1;
        }
        if (c == '*') {
            if (strlen(currentType) > 0)
                strcat(currentType, "*");
            return 1;
        }
        if (c == '[') {
            while ((c = getChar()) != ']');
            strcat(currentType, "[]");
            return 1;
        }
        if (isdigit(c)) {
            while (isdigit(c = getChar()));
            putBack(c);
            return 1;
        }
        if (c == ';' || c == '{') {
            currentType[0] = '\0';
            return 1;
        }
        if (isArithmetic(c)) return 1;
        if (isRelop(c)) return 1;
        if (isLogic(c)) return 1;
        if (isSpecial(c)) return 1;
    }
    return 0;
}

int main(){
    fp = fopen("in.c", "r");
    if (!fp) {
        printf("Cannot open file\n");
        return 0;
    }
    Initialize();
    while (getNextToken());
    fclose(fp);
    Display();
    return 0;
}
```

In.c:

```
#include <stdio.h>
#define PI 3.14

/* This is a multi-line
comment to test the lexer
*/

int main() {
    int radius = 10; // This is a single-line comment
    char msg = ‘Q’;
    float area;

    if (radius >= 10 && radius != 0) {
        area = PI * radius * radius;
        printf("Area is: %f", area);
    } else {
        return 0;
    }

    /* Testing logic and arithmetic */
    int x = (5 + 3) * 2;
    return 0;
}
```

Output:



```
6CSEC1@dbl-23:~/Documents/CD_230905152/L4$ cc -o abc 1a.c
6CSEC1@dbl-23:~/Documents/CD_230905152/L4$ ./abc

SYMBOL TABLE
LEXEME  TYPE    RETURN TYPE     SIZE
msg     char    -               1
x       int     -               4
area    float   -               4
radius  int     -               4
main    func    int             0
```