Lab 4

Name: Paawan Kohli

Reg No: 180905416

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

// global row and col tracker
int row = 1;
int col = 0;

char keyword[][20] = {"auto", "break", "case", "char", "const", "continue",
                      "default", "do", "double", "else", "enum", "extern",
                      "float", "for", "goto", "if", "int", "long", "register",
                      "return", "short", "signed", "sizeof", "static", "struct",
                      "switch", "typedef", "union", "unsigned", "void",
                      "volatile", "while", "printf", "scanf", "bool"
                     };

// no of keywords = 32 c keywords + printf + scanf + bool
int nok = 32 + 3;

char datatype[][20] = {"int", "char", "bool", "void"};

// no. of datatypes
int nod = 4;

char dbuff[20];

// when int, char, bool or void is encountered,
// an identifier is expected. Hence, expectID is set to 1
int expectID = 0;

char currTableName[20];

typedef struct {
        char token_name[50];
        unsigned int row, col;
        int index;
} token;

typedef struct {
        int index;
        char Lex_name[50];
        char type[20];
        int size;
} tableRow;

typedef struct {
        int entries;
        tableRow tR[50];
} table;

table currTable;

void reset_table() {
        currTable.entries = 0;
}

void print_table(FILE* out) {
        if (currTable.entries == 0) {
                return;
        }
```

```
        fprintf(out, "%s\n", currTableName);
        fprintf(out, "\tLex_Name\tType\tSize\n");

        for (int i = 0; i < currTable.entries; i++) {
                fprintf(out, "%d\t%s\t\t%s\t%d\n", currTable.tR[i].index,
                        currTable.tR[i].Lex_name, currTable.tR[i].type, currTable.tR[i].size);
        }

        fprintf(out, "\n");
}

token create_token(char n[], int r, int c, int i) {
        token m;

        strcpy(m.token_name, n);
        m.row = r;
        m.col = c;
        m.index = i;

        return m;
}

void insert(char n[], char t[], int s) {
        int m = currTable.entries;

        currTable.tR[m].index = m + 1;
        strcpy(currTable.tR[m].Lex_name, n);
        strcpy(currTable.tR[m].type, t);
        currTable.tR[m].size = s;

        currTable.entries++;
}

void print_token(token t, FILE* out) {
        fprintf(out, "<%s,%d,%d,%d>", t.token_name, t.row, t.col, t.index);
}

int search(char* name) {
        for (int i = 0; i < currTable.entries; i++)
                if (strcmp(name, currTable.tR[i].Lex_name) == 0)
                        return i + 1;

        return 0;
}

int getNextToken(FILE* f, FILE* out, FILE* st) {
        int cn;
        int t_index = -1;
        int c = getc(f);
        col++;

        if (c == EOF) {
                return 0;
        }

        token T;

        if (c == '\n') {
                c = getc(f);
                row++;
                col = 1;
                putc('\n', out);
        }

        // handle comments
        if (c == '/') {
                c = getc(f);

                // single line comments
                if (c == '/') {

                        do {
                                c = getc(f);
                        } while (c != '\n');
```

```
                col = 0;
        }
        // multiline comments
        else if (c == '*') {
                do {

                        do {
                                c = getc(f);
                        } while (c != '*');

                        c = getc(f);
                } while (c != '/');

                c = getc(f);
                col--;
        }
        // wasn't a comment
        else {
                fseek(f, -2, SEEK_CUR);
                c = getc(f);
        }
}

// handle string literals
if (c == '\"') {
        int length = 0;

        do {
                c = getc(f);
                length++;
        } while (c != '\"');

        T = create_token("string", row, col, -1);
        col += length;
}

// handle pre processor directives
else if (c == '#') {
        do {
                c = getc(f);
        } while (c != '\n');

        col = 0;
        return 1;
}

// handle ++, += and +
else if (c == '+') {
        c = getc(f);

        if (c == '+') {
                T = create_token("++", row, col, -1);
                col++;
        } else if (c == '=') {
                T = create_token("+=", row, col, -1);
                col++;
        } else {
                fseek(f, -1, SEEK_CUR);
                T = create_token("+", row, col, -1);
        }

}

// handle --, -+ and -
else if (c == '-') {
        c = getc(f);
        if (c == '-') {
                T = create_token("--", row, col, -1);
                col++;
        } else if (c == '=') {
                T = create_token("-=", row, col, -1);
                col++;
        } else {
                fseek(f, -1, SEEK_CUR);
                T = create_token("-", row, col, -1);
```

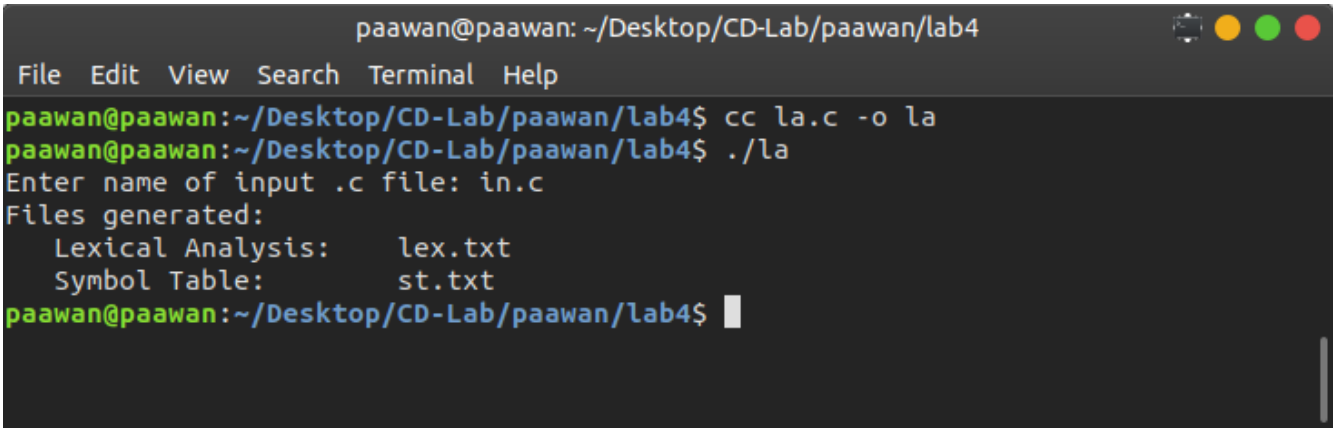
```
        }
    }

    // handle *= and *
    else if (c == '*') {
        c = getc(f);
        if (c == '=') {
            T = create_token("*=", row, col, -1);
            col++;
        } else {
            fseek(f, -1, SEEK_CUR);
            T = create_token("*", row, col, -1);
        }
    }

    // handle /= and /
    else if (c == '/') {
        c = getc(f);
        if (c == '=') {
            T = create_token("/=", row, col, -1);
            col++;
        } else {
            fseek(f, -1, SEEK_CUR);
            T = create_token("/", row, col, -1);
        }
    }

    // handle %= and %
    else if (c == '%') {
        c = getc(f);
        if (c == '=') {
            T = create_token("%=", row, col, -1);
            col++;
        } else {
            fseek(f, -1, SEEK_CUR);
            T = create_token("%", row, col, -1);
        }
    }

    // handle == and =
    else if (c == '=') {
        c = getc(f);
        if (c == '=') {
            T = create_token("==", row, col, -1);
            col++;
        } else {
            fseek(f, -1, SEEK_CUR);
            T = create_token("=", row, col, -1);
        }
    }

    // handle != and !
    else if (c == '!') {
        c = getc(f);
        if (c == '=') {
            T = create_token("!=", row, col, -1);
            col++;
        } else {
            fseek(f, -1, SEEK_CUR);
            T = create_token("!", row, col, -1);
        }
    }

    // handle >= and >
    else if (c == '>') {
        c = getc(f);
        if (c == '=') {
            T = create_token("GE", row, col, -1);
            col++;
        } else {
            fseek(f, -1, SEEK_CUR);
            T = create_token("GT", row, col, -1);
        }
    }
```

```
        // handle <= and <
        else if (c == '<') {
                c = getc(f);
                if (c == '=') {
                        T = create_token("LE", row, col, -1);
                        col++;
                } else {
                        fseek(f, -1, SEEK_CUR);
                        T = create_token("LT", row, col, -1);
                }
        }

        // handle && and &
        else if (c == '&') {
                c = getc(f);
                if (c == '&') {
                        T = create_token("&&", row, col, -1);
                        col++;
                } else {
                        fseek(f, -1, SEEK_CUR);
                        T = create_token("&", row, col, -1);
                }
        }

        // handle || and |
        else if (c == '|') {
                c = getc(f);
                if (c == '|') {
                        T = create_token("||", row, col, -1);
                        col++;
                } else {
                        fseek(f, -1, SEEK_CUR);
                        T = create_token("|", row, col, -1);
                }
        }

        // handle keywords
        else if (isalpha(c)) {
                int length = 0;
                int flag = 0;
                char buffer[20];

                do {
                        buffer[length++] = c;
                        c = getc(f);
                } while (isalpha(c));

                buffer[length] = '\0';


                // see if the word in buffer was a keyword. If yes, set flag to 1
                for (int j = 0; j < nok; j++) {
                        if (strcmp(keyword[j], buffer) == 0) {
                                for (int k = 0; k < nod; k++) {
                                        if (strcmp(datatype[k], buffer) == 0) {
                                                strcpy(dbuff, buffer);
                                                expectID = 1;
                                        }
                                }

                                T = create_token(buffer, row, col, t_index);
                                flag++;
                                break;
                        }
                }

                // wasn't a keyword
                if (flag == 0) {
                        fseek(f, -1, SEEK_CUR);
                        char cnext = getc(f);

                        if (expectID == 1 && cnext == '(') {
                                print_table(st);
                                reset_table();
                                strcpy(currTableName, buffer);
```

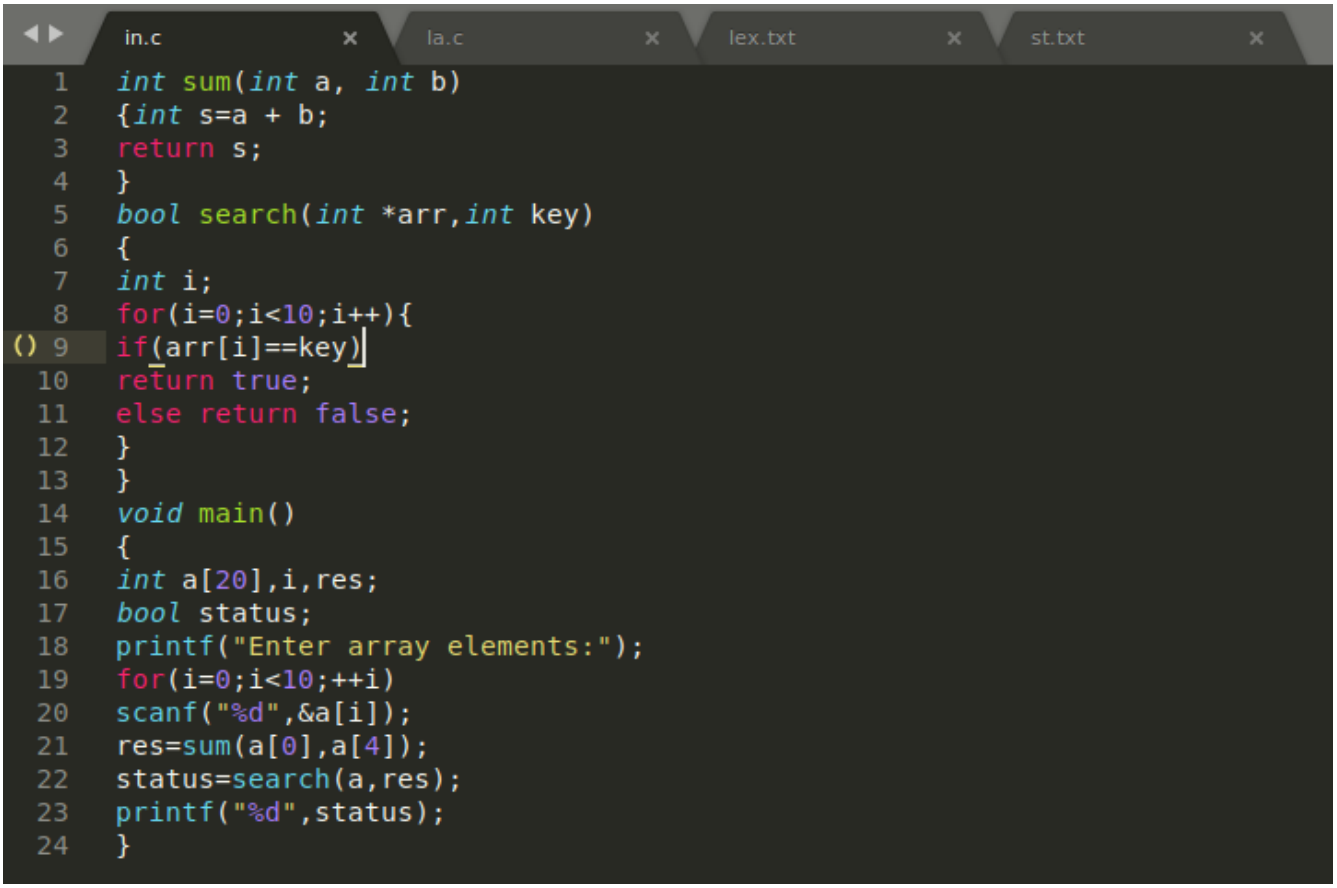
```
                        t_index = 0;
                } else if (expectID == 1) {
                        int t_size;

                        switch (dbuff[0]) {
                        case 'i':
                                t_size = 4;
                                break;
                        case 'c':
                                t_size = 1;
                                break;
                        case 'v':
                                t_size = 4;
                                break;
                        case 'b':
                                t_size = 1;
                                break;
                        }

                        if (cnext == '[') {
                                int elements = 0;
                                int extracount = 1;
                                cnext = getc(f);

                                while (cnext != ']') {
                                        elements *= 10;
                                        elements += (cnext - 48);
                                        cnext = getc(f);
                                        extracount++;
                                }

                                fseek(f, -1 * extracount, SEEK_CUR);
                                t_size *= elements;
                        }

                        insert(buffer, dbuff, t_size);
                        t_index = currTable.entries;
                }
                else if (cnext == '(') {

                        if (search(buffer) == 0) {
                                insert(buffer, "func", -1);
                                t_index = currTable.entries;
                        } else {
                                t_index = search(buffer);
                        }

                }
                else {
                        t_index = search(buffer);
                }

                fseek(f, -1, SEEK_CUR);
                length--;
                T = create_token("id", row, col, t_index);
        }
        col += length;
} else if (isdigit(c) || c == '.') {
        int length = 0;

        do {
                length++;
                c = getc(f);
        } while (isdigit(c) || c == '.');

        T = create_token("num", row, col, -1);
        col += length - 1;
        fseek(f, -1, SEEK_CUR);
} else {
        char temp[2]; temp[0] = c; temp[1] = '\0';
        T = create_token(temp, row, col, -1);
}

if (c == ';' || c == '(' || c == '=' || c == ')') {
        expectID = 0;
```

```
        }

        if (T.token_name[0] != '\n' && T.token_name[0] != ' ' && T.token_name[0] != '\t') {
                print_token(T, out);
        }

        return 1;
}

int main() {
        currTable.entries = 0;

        char filename[20];
        printf("Enter name of input .c file: ");
        scanf("%s", filename);

        FILE* f = fopen(filename, "r");
        FILE* lex = fopen("lex.txt", "w");
        FILE* st = fopen("st.txt", "w");

        while (getNextToken(f, lex, st));

        print_table(st);

        fclose(f);
        fclose(lex);
        fclose(st);

        printf("Files generated:\n");
        printf("   Lexical Analysis:\tlex.txt\n");
        printf("   Symbol Table:\tst.txt\n");


        return 0;
}
```

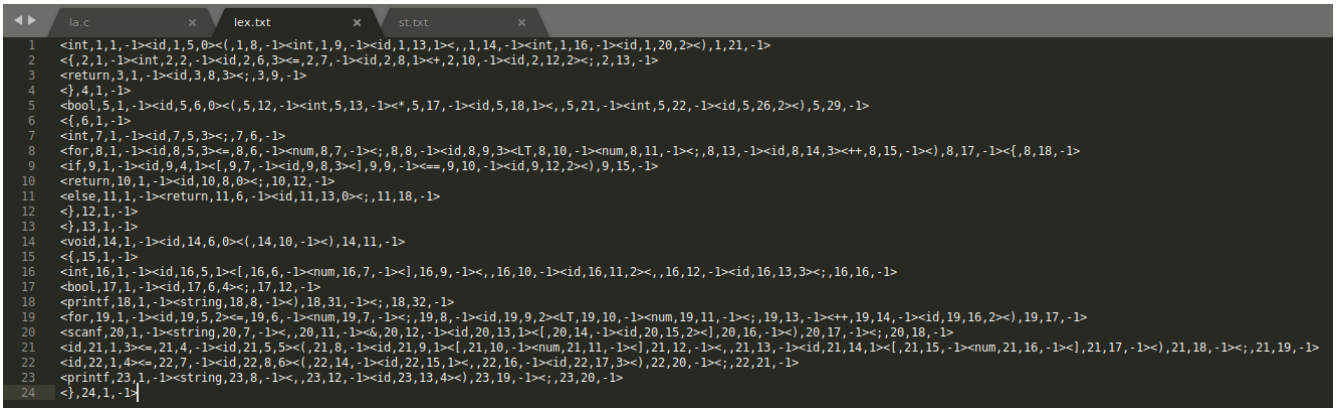
Terminal Output:



Input file:



Lexical Analysis:



Symbol Table: