

# Lab 4

Name: Paawan Kohli

Reg No: 180905416

```
#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(">=", 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("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:

```
paawan@paawan: ~/Desktop/CD-Lab/paawan/lab4
File Edit View Search Terminal Help
paawan@paawan:~/Desktop/CD-Lab/paawan/lab4$ cc la.c -o la
paawan@paawan:~/Desktop/CD-Lab/paawan/lab4$ ./la
Enter name of input .c file: in.c
Files generated:
  Lexical Analysis:    lex.txt
  Symbol Table:        st.txt
paawan@paawan:~/Desktop/CD-Lab/paawan/lab4$
```

Input file:

```
in.c  la.c  lex.txt  st.txt
1  int sum(int a, int b)
2  {int s=a + b;
3  return s;
4  }
5  bool search(int *arr,int key)
6  {
7  int i;
8  for(i=0;i<10;i++){
9  if(arr[i]==key)
10 return true;
11 else return false;
12 }
13 }
14 void main()
15 {
16 int a[20],i,res;
17 bool status;
18 printf("Enter array elements:");
19 for(i=0;i<10;++i)
20 scanf("%d",&a[i]);
21 res=sum(a[0],a[4]);
22 status=search(a,res);
23 printf("%d",status);
24 }
```



## Lexical Analysis:

```
1 <int,1,1,-1><id,1,5,0><{,1,8,-1><int,1,9,-1><id,1,13,1><,,1,14,-1><int,1,16,-1><id,1,20,2><},1,21,-1>
2 <[,2,1,-1><int,2,2,-1><id,2,6,3><=,2,7,-1><id,2,8,1><+,2,10,-1><id,2,12,2><,,2,13,-1>
3 <return,3,1,-1><id,3,8,3><;,3,9,-1>
4 <},4,1,-1>
5 <bool,5,1,-1><id,5,6,0><{,5,12,-1><int,5,13,-1><*,5,17,-1><id,5,18,1><,,5,21,-1><int,5,22,-1><id,5,26,2><},5,29,-1>
6 <[,6,1,-1>
7 <int,7,1,-1><id,7,5,3><;,7,6,-1>
8 <for,8,1,-1><id,8,5,3><=,8,6,-1><num,8,7,-1><;,8,8,-1><id,8,9,3><LT,8,10,-1><num,8,11,-1><;,8,13,-1><id,8,14,3><++,8,15,-1><},8,17,-1><{,8,18,-1>
9 <if,9,1,-1><id,9,4,1><[,9,7,-1><id,9,8,3><[,9,9,-1><=,9,10,-1><id,9,12,2><},9,15,-1>
10 <return,10,1,-1><id,10,8,0><;,10,12,-1>
11 <else,11,1,-1><return,11,6,-1><id,11,13,0><;,11,18,-1>
12 <},12,1,-1>
13 <},13,1,-1>
14 <void,14,1,-1><id,14,6,0><{,14,10,-1><},14,11,-1>
15 <[,15,1,-1>
16 <int,16,1,-1><id,16,5,1><[,16,6,-1><num,16,7,-1><[,16,9,-1><,,16,10,-1><id,16,11,2><,,16,12,-1><id,16,13,3><;,16,16,-1>
17 <bool,17,1,-1><id,17,6,4><;,17,12,-1>
18 <printf,18,1,-1><string,18,8,-1><[,18,31,-1><;,18,32,-1>
19 <for,19,1,-1><id,19,5,2><=,19,6,-1><num,19,7,-1><;,19,8,-1><id,19,9,2><LT,19,10,-1><num,19,11,-1><;,19,13,-1><++,19,14,-1><id,19,16,2><},19,17,-1>
20 <scanf,20,1,-1><string,20,7,-1><,,20,11,-1><[,20,12,-1><id,20,13,1><[,20,14,-1><id,20,15,2><[,20,16,-1><},20,17,-1><;,20,18,-1>
21 <id,21,1,3><=,21,4,-1><id,21,5,5><{,21,8,-1><id,21,9,1><[,21,10,-1><num,21,11,-1><[,21,12,-1><;,21,13,-1><id,21,14,1><[,21,15,-1><num,21,16,-1><[,21,17,-1><},21,18,-1><;,21,19,-1>
22 <id,22,1,4><=,22,7,-1><id,22,8,6><{,22,14,-1><id,22,15,1><;,22,16,-1><id,22,17,3><},22,20,-1><;,22,21,-1>
23 <printf,23,1,-1><string,23,8,-1><;,23,12,-1><id,23,13,4><},23,19,-1><;,23,20,-1>
24 <},24,1,-1>
```

## Symbol Table:

```
1 sum
2 Lex_Name Type Size
3 1 a int 4
4 2 b int 4
5 3 s int 4
6
7 search
8 Lex_Name Type Size
9 1 arr int 4
10 2 key int 4
11 3 i int 4
12
13 main
14 Lex_Name Type Size
15 1 a int 80
16 2 i int 4
17 3 res int 4
18 4 status bool 1
19 5 sum func -1
20 6 search func -1
21
```