Name: Sapthami Upadhya
Section: CSE A
Roll No.: 15
Reg. No.: 230905090

Week 3

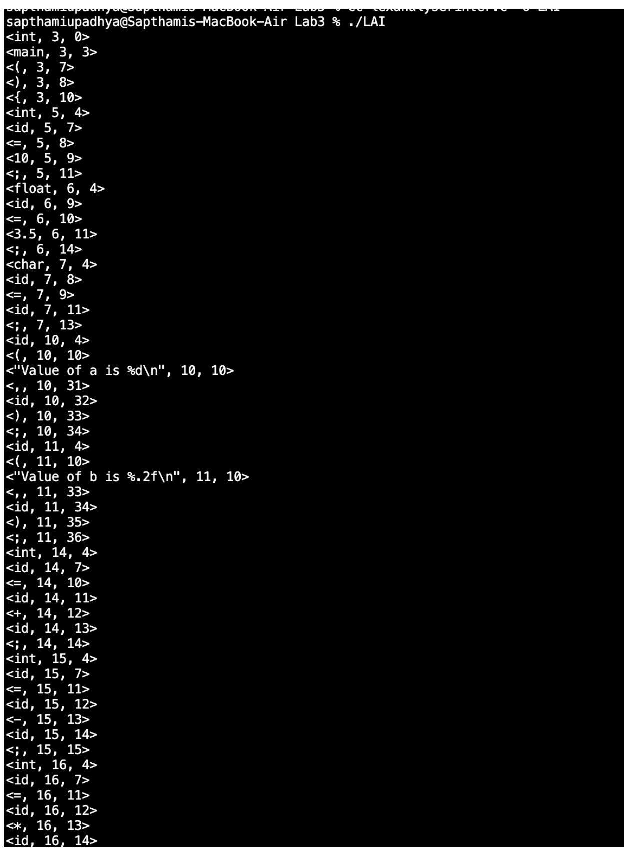
2. Design a lexical analyzer that includes a getNextToken() function for processing a simple C program. The analyzer should construct a token structure containing the row number, column number, and token type for each identified token. The getNextToken() function must ignore tokens located within singleline or multi-line comments, as well as those found inside string literals. Additionally, it should strip out preprocessor directives.

Code:

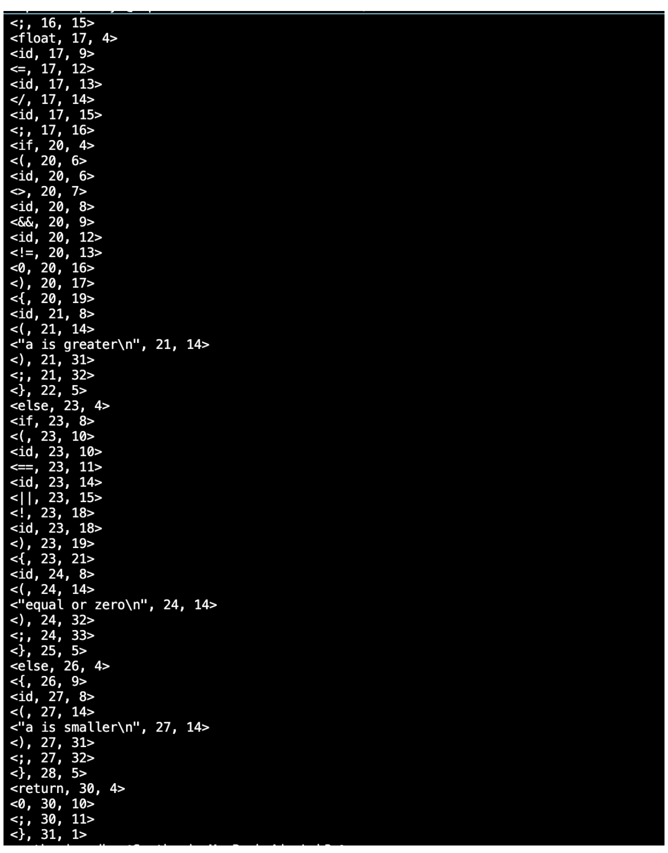
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
typedef struct{
    char type[1000];
    int row, col;
}TOKEN;
char* keywords[] = {"int", "float", "char", "void", "if", "else", "while",
"for", "return", "main"};
int keycount = 10;
int row, col;
FILE* fp;
int isKeyword(char* str){
    for(int i = 0; i < keycount; i++)
        if(strcmp(str, keywords[i]) == 0) return 1;
    return 0;
}
TOKEN readString(){
    TOKEN token;
    int startrow = row;
    int startcol = col;
    int i = 0;
    char buf[256];
    char c = getc(fp);
    col++;
    buf[i++] = c;
    while((c = fgetc(fp)) != EOF){
        col++;
        buf[i++] = c;
        if(c == '"') break;
        if(c == '\n'){
            row++;
            col = 0;
        }
    }
```

```
        buf[i] = '\0';
        token.row = startrow;
        token.col = startcol;
        strcpy(token.type, buf);
        return token;
    }
    TOKEN readId(){
        TOKEN token;
        int startrow = row;
        int startcol = col;
        char buf[100];
        int i = 0;
        char c = getc(fp);
        col++;
        buf[i++] = c;
        while(1){
            char next = getc(fp);
            if(isalnum(next) || next == '_'){
                col++;
                buf[i++] = next;
            }
            else{
                fseek(fp, -1, SEEK_CUR);
                col--;
                break;
            }
        }
        buf[i] = '\0';
        token.row = startrow;
        token.col = startcol;
        if(isKeyword(buf)) strcpy(token.type, buf);
        else strcpy(token.type, "id");
        return token;
    }
    TOKEN readNum(){
        TOKEN token;
        int startrow = row;
        int startcol = col;
        char buf[100];
        int i = 0;
        int decimal = 0;
        char c = getc(fp);
        col++;
        buf[i++] = c;
        while(1){
            char next = getc(fp);
            if(isdigit(next)){
                col++;
                buf[i++] = next;
            }
            else if(next == '.' && !decimal){
                col++;
                decimal = 1;
                buf[i++] = next;
            }
            else{
                fseek(fp, -1, SEEK_CUR);
                col--;
                break;
            }
        }
        buf[i] = '\0';
        token.row = startrow;
        token.col = startcol;
        strcpy(token.type, buf);
        return token;
    }
    TOKEN readOp(){
        TOKEN token;
        int startrow = row;
        int startcol = col;
        char buf[10];
        int i = 0;
        char c = getc(fp);
        col++;
        buf[i++] = c;
        char next = getc(fp);
        if((c=='=' && next=='=') || (c=='<' && next=='=') || (c=='>' &&
    next=='=') || (c=='!' && next=='=') || (c=='&' && next=='&') || (c=='|' &&
    next=='|')){
            col++;
            buf[i++] = next;
        }
        else {
            fseek(fp, -1, SEEK_CUR);
            col--;
        }
        buf[i] = '\0';
        token.row = startrow;
        token.col = startcol;
        strcpy(token.type, buf);
        return token;
    }
    TOKEN readSymbol(){
        TOKEN token;
        char c = getc(fp);
        col++;
        token.row = row;
        token.col = col;
        token.type[0] = c;
        token.type[1] = '\0';
        return token;
    }
    TOKEN getNextToken(){
        TOKEN token;
        char c;
        c = getc(fp);
        if(c=='\n'){
            row++;
            col = 0;
            return getNextToken();
        }
        if(c == ' ' && c == '\t'){
            col++;
            return getNextToken();
        }
        if(c == EOF){
            strcpy(token.type, "EOF");
            return token;
        }
        fseek(fp, -1, SEEK_CUR);
        if(c == '"') return readString();
        if(isalpha(c) || c == '_') return readId();
        if(isdigit(c)) return readNum();
        if(c=='+' || c=='-' || c=='*' || c=='/' || c=='<'
        || c=='>' || c=='=' || c=='!' || c=='&' || c=='|') return readOp();
        if(c==';' || c==',' || c=='(' || c==')'
        || c=='{' || c=='}' || c=='[' || c==']') return readSymbol();
        getc(fp);
        col++;
        return getNextToken();
    }
    void preprocess(FILE* src, FILE* dest){
        char c, next;
        while((c=getc(src)) != EOF){
            //preprocessor directive
            if(c == '#'){
                while( c != '\n' && c != EOF)
                    c = getc(src);
                putc('\n', dest);
                continue;
            }
            //comment
            if(c == '/'){
                next = getc(src);
                if(next == '/'){
                    while(c != '\n' && c != EOF)
                        c = getc(src);
                    putc('\n', dest);
                    continue;
                }
                else if(next == '*'){
                    while(1){
                        c = getc(src);
                        if(c == '*'){
                            char end = getc(src);
                            if(end == '/') break;
                        }
                    }
                    continue;
                }
                else{
                    putc(c, dest);
                    putc(next, dest);
                    continue;
                }
            }
            putc(c, dest);
        }
    }
    int main(){
        FILE* src, *inter;
        src = fopen("samplela.c", "r");
        if(!src){
            printf("Cannot open source file\n");
            return 0;
        }
        inter = fopen("inter.c", "w");
        preprocess(src, inter);
        fclose(src);
        fclose(inter);

        fp = fopen("inter.c", "r");
        if(!fp){
            printf("Cannot open intermediate file\n");
            return 0;
        }
        TOKEN token;
        row = 1;
        col = 0;
        do{
            token = getNextToken();
            if(strcmp(token.type, "EOF") != 0)
                printf("<%s, %d, %d>\n", token.type, token.row, token.col);
        } while(strcmp(token.type, "EOF") != 0);
        fclose(fp);
        return 0;
    }
```

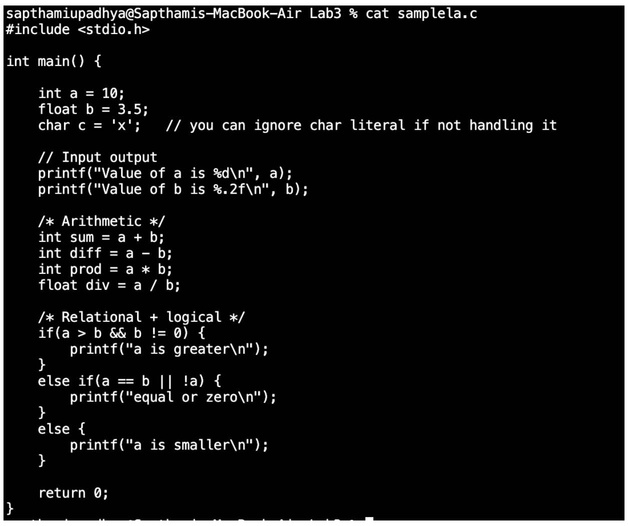
Input/Output:



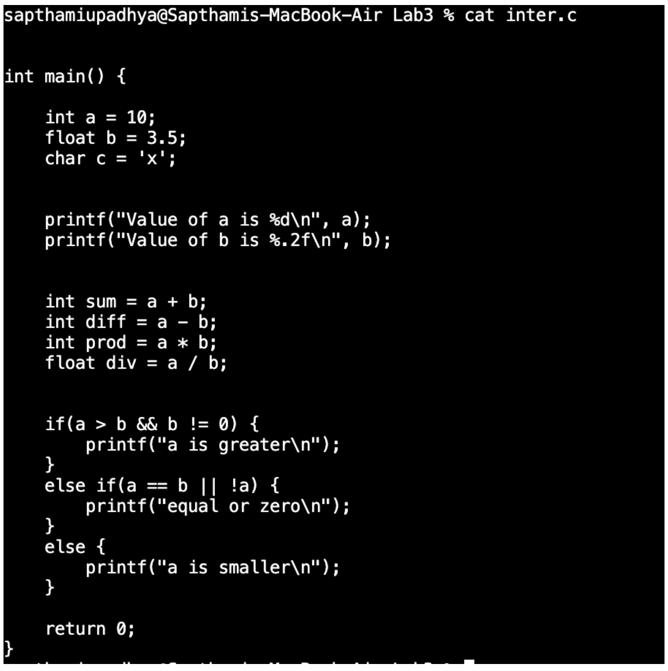
```
sapthamiupadhya@Sapthamis-MacBook-Air Lab3 % ./LAI
<int, 3, 0>
<main, 3, 3>
<(, 3, 7>
<), 3, 8>
<{, 3, 10>
<int, 5, 4>
<id, 5, 7>
<=, 5, 8>
<10, 5, 9>
<;, 5, 11>
<float, 6, 4>
<id, 6, 9>
<=, 6, 10>
<3.5, 6, 11>
<;, 6, 14>
<char, 7, 4>
<id, 7, 8>
<=, 7, 9>
<id, 7, 11>
<;, 7, 13>
<id, 10, 4>
<(, 10, 10>
<"Value of a is %d\n", 10, 10>
<,, 10, 31>
<id, 10, 32>
<), 10, 33>
<;, 10, 34>
<id, 11, 4>
<(, 11, 10>
<"Value of b is %.2f\n", 11, 10>
<,, 11, 33>
<id, 11, 34>
<), 11, 35>
<;, 11, 36>
<int, 14, 4>
<id, 14, 7>
<=, 14, 10>
<id, 14, 11>
<+, 14, 12>
<id, 14, 13>
<;, 14, 14>
<int, 15, 4>
<id, 15, 7>
<=, 15, 11>
<id, 15, 12>
<-, 15, 13>
<id, 15, 14>
<;, 15, 15>
<int, 16, 4>
<id, 16, 7>
<=, 16, 11>
<id, 16, 12>
<*, 16, 13>
<id, 16, 14>
```



```
<;, 16, 15>
<float, 17, 4>
<id, 17, 9>
<=, 17, 12>
<id, 17, 13>
</, 17, 14>
<id, 17, 15>
<;, 17, 16>
<if, 20, 4>
<(, 20, 6>
<id, 20, 6>
<>, 20, 7>
<id, 20, 8>
<&&, 20, 9>
<id, 20, 12>
<!=, 20, 13>
<0, 20, 16>
<), 20, 17>
<{, 20, 19>
<id, 21, 8>
<(, 21, 14>
<"a is greater\n", 21, 14>
<), 21, 31>
<;, 21, 32>
<}, 22, 5>
<else, 23, 4>
<if, 23, 8>
<(, 23, 10>
<id, 23, 10>
<==, 23, 11>
<id, 23, 14>
<||, 23, 15>
<!, 23, 18>
<id, 23, 18>
<), 23, 19>
<{, 23, 21>
<id, 24, 8>
<(, 24, 14>
<"equal or zero\n", 24, 14>
<), 24, 32>
<;, 24, 33>
<}, 25, 5>
<else, 26, 4>
<{, 26, 9>
<id, 27, 8>
<(, 27, 14>
<"a is smaller\n", 27, 14>
<), 27, 31>
<;, 27, 32>
<}, 28, 5>
<return, 30, 4>
<0, 30, 10>
<;, 30, 11>
<}, 31, 1>
```



```
sapthamiupadhya@Sapthamis-MacBook-Air Lab3 % cat samplela.c
#include <stdio.h>

int main() {

    int a = 10;
    float b = 3.5;
    char c = 'x';   // you can ignore char literal if not handling it

    // Input output
    printf("Value of a is %d\n", a);
    printf("Value of b is %.2f\n", b);

    /* Arithmetic */
    int sum = a + b;
    int diff = a - b;
    int prod = a * b;
    float div = a / b;

    /* Relational + logical */
    if(a > b && b != 0) {
        printf("a is greater\n");
    }
    else if(a == b || !a) {
        printf("equal or zero\n");
    }
    else {
        printf("a is smaller\n");
    }

    return 0;
}
```



```
sapthamiupadhya@Sapthamis-MacBook-Air Lab3 % cat inter.c


int main() {

    int a = 10;
    float b = 3.5;
    char c = 'x';


    printf("Value of a is %d\n", a);
    printf("Value of b is %.2f\n", b);


    int sum = a + b;
    int diff = a - b;
    int prod = a * b;
    float div = a / b;


    if(a > b && b != 0) {
        printf("a is greater\n");
    }
    else if(a == b || !a) {
        printf("equal or zero\n");
    }
    else {
        printf("a is smaller\n");
    }

    return 0;
}
```