CD Lab 3

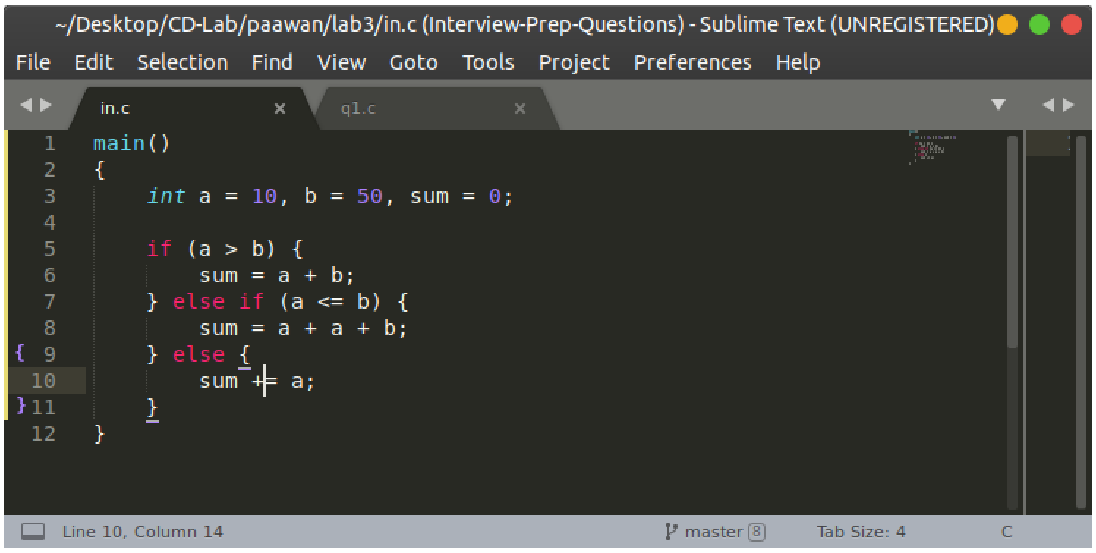
Name: Paawan Kohli

Reg No: 180905416

Sample: Identification of arithmetic and relational operators from the given input file

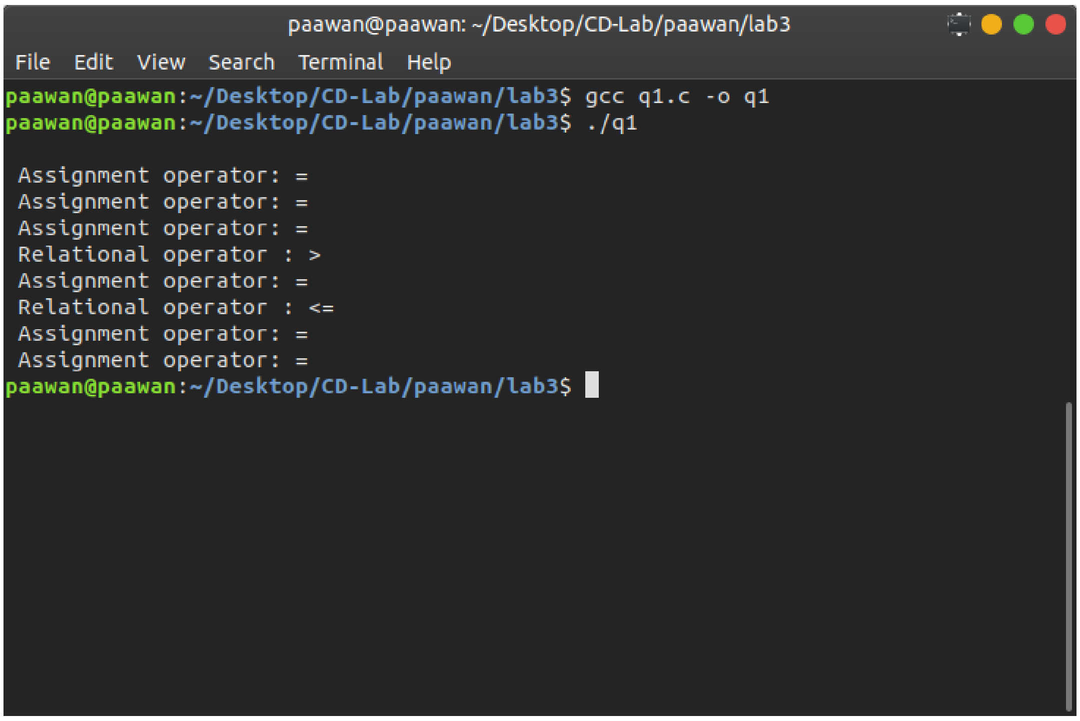
```c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

void main() {
        FILE *fp = fopen("in.c", "r");

        if (fp == NULL) {
                printf("Cannot open file \n");
                exit(0);
        }

        char c = fgetc(fp);
        char buf[30];

        while (c != EOF) {
                int i = 0;
                buf[0] = '\0';

                if (c == '=') {
                        buf[i++] = c;
                        c = fgetc(fp);

                        if (c == '=') {
                                buf[i++] = c;
                                buf[i] = '\0';
                                printf("\n Relational operator : %s", buf);
                        } else {
                                buf[i] = '\0';
                                printf("\n Assignment operator: %s", buf);
                        }

                } else {
                        if (c == '<' || c == '>' || c == '!') {
                                buf[i++] = c;
                                c = fgetc(fp);

                                if (c == '=') {
                                        buf[i++] = c;
                                }

                                buf[i] = '\0';
                                printf("\n Relational operator : %s", buf);
                        } else {
                                buf[i] = '\0';
                        }
                }

                c = fgetc(fp);
        }
        printf("\n");
}
```

Input file:


~/Desktop/CD-Lab/paawan/lab3/in.c (Interview-Prep-Questions) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
in.c
q1.c
Line 10, Column 14
master 8
Tab Size: 4
C

```
main()
{
    int a = 10, b = 50, sum = 0;

    if (a > b) {
        sum = a + b;
    } else if (a <= b) {
        sum = a + a + b;
    } else {
        sum += a;
    }
}
```

Terminal:


paawan@paawan: ~/Desktop/CD-Lab/paawan/lab3
File Edit View Search Terminal Help

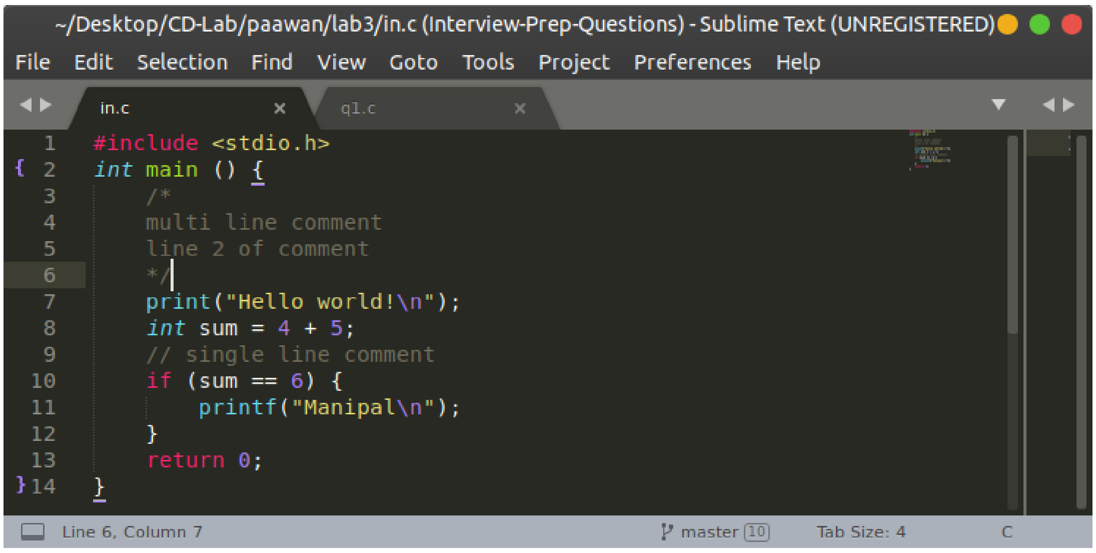
```
paawan@paawan:~/Desktop/CD-Lab/paawan/lab3$ gcc q1.c -o q1
paawan@paawan:~/Desktop/CD-Lab/paawan/lab3$ ./q1

 Assignment operator: =
 Assignment operator: =
 Assignment operator: =
 Relational operator : >
 Assignment operator: =
 Relational operator : <=
 Assignment operator: =
 Assignment operator: =
paawan@paawan:~/Desktop/CD-Lab/paawan/lab3$ 
```

Design a lexical analyzer which contains getNextToken( ) for a simple C program to create a structure of token each time and return, which includes row number, column number and token type. The tokens to be identified are arithmetic operators, relational operators, logical operators, special symbols, keywords, numerical constants, string literals and identifiers. Also, getNextToken() should ignore all the tokens when encountered inside single line or multiline comment or inside string literal. Preprocessor directive should also be stripped.

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

struct token {
        char name[30];
        int row, col;
};

char splChars[] = {':', ';', '(', ')', '{', '}', ',', '.', '?'};
char aritChars[] = {'+', '-', '*', '/'};

char keywords[][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"
                      };

int row = 1, col = 0, ca, cb;

struct token getNextToken(FILE *in) {
        char buffer[50];

        while (ca != EOF) {

                // handle preprocessors
                if (ca == '#') {
                        while (ca != '\n') {
                                col++;
                                ca = getc(in);
                        }
                }

                // handle comments
                if (ca == '/') {
                        col++;
                        cb = getc(in);
                        if (cb == '/') {
                                while (ca != '\n') {
                                        col++;
                                        ca = getc(in);
                                }
                        } else if (cb == '*') {
                                do {
                                        while (ca != '*') {
                                                col++;

                                                if (ca == '\n') {
                                                        col = 1;
                                                        row++;
                                                }

                                                ca = getc(in);
                                        }

                                        ca = getc(in);
                                } while (ca != '/');
                                ca = getc(in);

                        }
                }

                // Literal
                if (ca == '"') {

                        int i = 0;
                        ca = getc(in);
                        col++;

                        while (ca != '"') {
                                buffer[i++] = ca;
                                ca = getc(in);
                        }

                        buffer[i] = '\0';
                        struct token t;
                        int j = 0;

                        while (buffer[j] != '\0') {
                                t.name[j] = buffer[j];
                                j++;
                        }

                        t.name[j] = '\0';
                        t.row = row;
                        t.col = col;
                        col = col + strlen(buffer);
                        ca = getc(in);

                        return t;
                }

                //blank space
                if (ca == ' ') {
                        col++;
                        ca = getc(in);
                }

                // handle new line char
                if (ca == '\n') {
                        row++;
                        col = 1;
                        ca = getc(in);
                        printf("\n");
                }

                // handle spl char
                for (int i = 0; i < 9; i++) {
                        if (ca == splChars[i]) {
                                struct token t;
                                t.name[0] = ca;
                                t.name[1] = '\0';
                                t.row = row;
                                t.col = col;
                                ca = getc(in);
                                col++;
                                return t;
                        }
                }

                // handle arithmatic operators
                for (int i = 0; i < 4; i++) {
                        if (ca == aritChars[i]) {
                                struct token t;
                                t.name[0] = ca;
                                t.name[1] = '\0';
                                t.row = row;
                                t.col = col;
                                ca = getc(in);
                                col++;
                                return t;
                        }
                }

                // handle assignment or equals operator
                if (ca == '=') {
                        ca = getc(in);
                        if (ca == '=') {
                                struct token t;
                                t.name[0] = '=';
                                t.name[1] = '=';
                                t.name[2] = '\0';
                                t.row = row;
                                t.col = col;
                                ca = getc(in);
                                col += 2;
                                return t;
                        } else {
                                struct token t;
                                t.name[0] = '=';
                                t.name[1] = '\0';
                                t.row = row;
                                t.col = col;
                                col++;
                                return t;
                        }
                }

                // handle logical ops
                if (ca == '|') {
                        ca = getc(in);
                        if (ca == '|') {
                                struct token t;
                                t.name[0] = '|';
                                t.name[1] = '|';
                                t.name[2] = '\0';
                                t.row = row;
                                t.col = col;
                                ca = getc(in);
                                col += 2;
                                return t;
                        } else {
                                struct token t;
                                t.name[0] = '|';
                                t.name[1] = '\0';
                                t.row = row;
                                t.col = col;
                                col++;
                                return t;
                        }
                } else if (ca == '&') {
                        ca = getc(in);
                        if (ca == '&') {
                                struct token t;
                                t.name[0] = '&';
                                t.name[1] = '&';
                                t.name[2] = '\0';
                                t.row = row;
                                t.col = col;
                                ca = getc(in);
                                col += 2;
                                return t;
                        } else {
                                struct token t;
                                t.name[0] = '&';
                                t.name[1] = '\0';
                                t.row = row;
                                t.col = col;
                                col++;
                                return t;
                        }
                } else if (ca == '^') {
                        struct token t;
                        t.name[0] = '^';
                        t.name[1] = '\0';
                        t.row = row;
                        t.col = col;
                        col++;
                        return t;
                }

                // handle relational operators
                if (ca == '<') {
                        ca = getc(in);
                        if (ca == '=') {
                                struct token t;
                                t.name[0] = '<';
                                t.name[1] = '=';
                                t.name[2] = '\0';
                                t.row = row;
                                t.col = col;
                                ca = getc(in);
                                col += 2;
                                return t;
                        } else {
                                struct token t;
                                t.name[0] = '<';
                                t.name[1] = '\0';
                                t.row = row;
                                t.col = col;
                                col++;
                                return t;
                        }
                } else if (ca == '>') {
                        ca = getc(in);
                        if (ca == '=')
                        {
                                struct token t;
                                t.name[0] = '>';
                                t.name[1] = '=';
                                t.name[2] = '\0';
                                t.row = row;
                                t.col = col;
                                ca = getc(in);
                                col += 2;
                                return t;
                        }
                        else {
                                struct token t;
                                t.name[0] = '>';
                                t.name[1] = '\0';
                                t.row = row;
                                t.col = col;
                                col++;
                                return t;
                        }
                }

                // handle numerics
                int i = 0;
                if (isdigit(ca)) {

                        while (isdigit(ca)) {
                                buffer[i++] = ca;
                                ca = getc(in);
                        }

                        buffer[i] = '\0';
                        struct token t;
                        strcpy(t.name, buffer);
                        t.row = row;
                        t.col = col;
                        col += strlen(buffer);
                        return t;
                }

                // handle keywords
                i = 0;
                while (isalpha(ca)) {
                        buffer[i++] = ca;
                        ca = getc(in);
                }

                buffer[i] = '\0';

                for (int j = 0; j < 13; j++) {
                        if (strcmp(buffer, keywords[j]) == 0) {
                                struct token t;
                                strcpy(t.name, buffer);
                                t.row = row;
                                t.col = col;
                                col = col + strlen(buffer);
                                return t;
                        }
                }

                if (buffer[0] != '\0') {
                        struct token t;
                        strcpy(t.name, buffer);
                        t.row = row;
                        t.col = col;
                        col += strlen(buffer);
                        return t;
                }

                ca = getc(in);
                col++;
        }

        struct token t;
        t.row = -1;
}
```

```
void main() {
        FILE* in = fopen("in.c", "r");

        if (in == NULL) {
                printf("Cannot open file \n");
                exit(0);
        }

        ca = getc(in);
        col = 1;

        while (ca != EOF) {
                struct token t = getNextToken(in);

                if (t.row == -1) {
                        break;
                }

                printf("<%s,%d,%d>", t.name, t.row, t.col );
        }

        printf("\n");
}
```

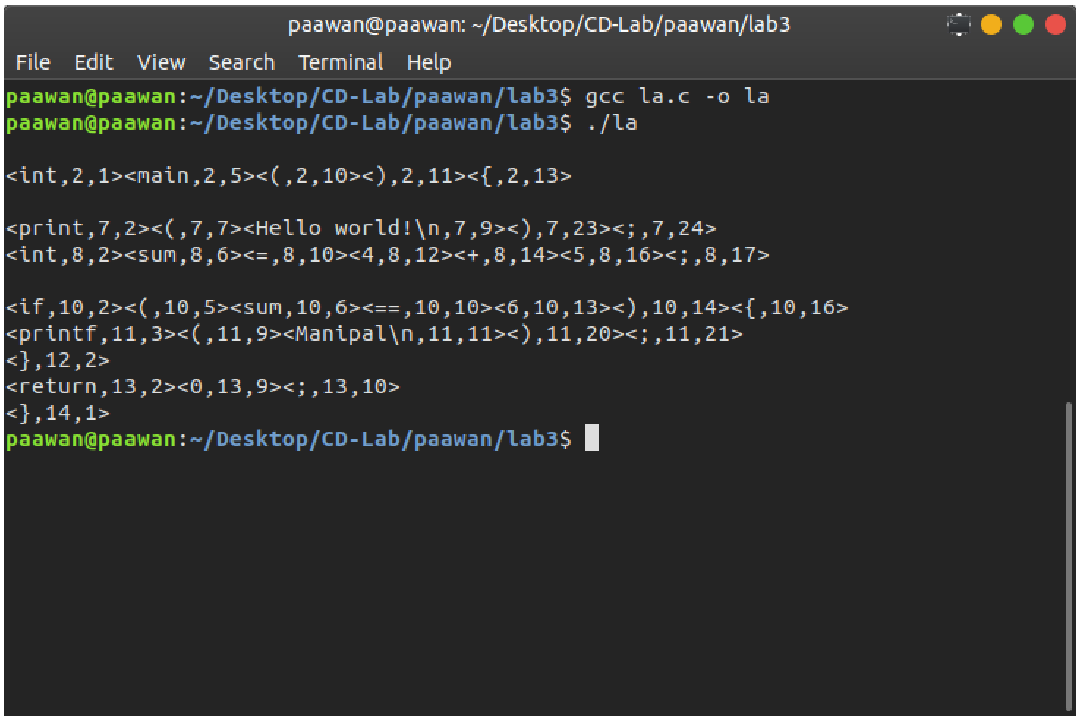
Input file:



~/Desktop/CD-Lab/paawan/lab3/in.c (Interview-Prep-Questions) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
in.c
q1.c
Line 6, Column 7
master 10
Tab Size: 4
C

```
#include <stdio.h>
int main () {
    /*
    multi line comment
    line 2 of comment
    */
    print("Hello world!\n");
    int sum = 4 + 5;
    // single line comment
    if (sum == 6) {
        printf("Manipal\n");
    }
    return 0;
}
```

Terminal:


paawan@paawan: ~/Desktop/CD-Lab/paawan/lab3

```
paawan@paawan:~/Desktop/CD-Lab/paawan/lab3$ gcc la.c -o la
paawan@paawan:~/Desktop/CD-Lab/paawan/lab3$ ./la

<int,2,1><main,2,5><(,2,10><),2,11><{,2,13>

<print,7,2><(,7,7><Hello world!\n,7,9><),7,23><;,7,24>
<int,8,2><sum,8,6><=,8,10><4,8,12><+,8,14><5,8,16><;,8,17>

<if,10,2><(,10,5><sum,10,6><==,10,10><6,10,13><),10,14><{,10,16>
<printf,11,3><(,11,9><Manipal\n,11,11><),11,20><;,11,21>
<},12,2>
<return,13,2><0,13,9><;,13,10>
<},14,1>
paawan@paawan:~/Desktop/CD-Lab/paawan/lab3$ 
```