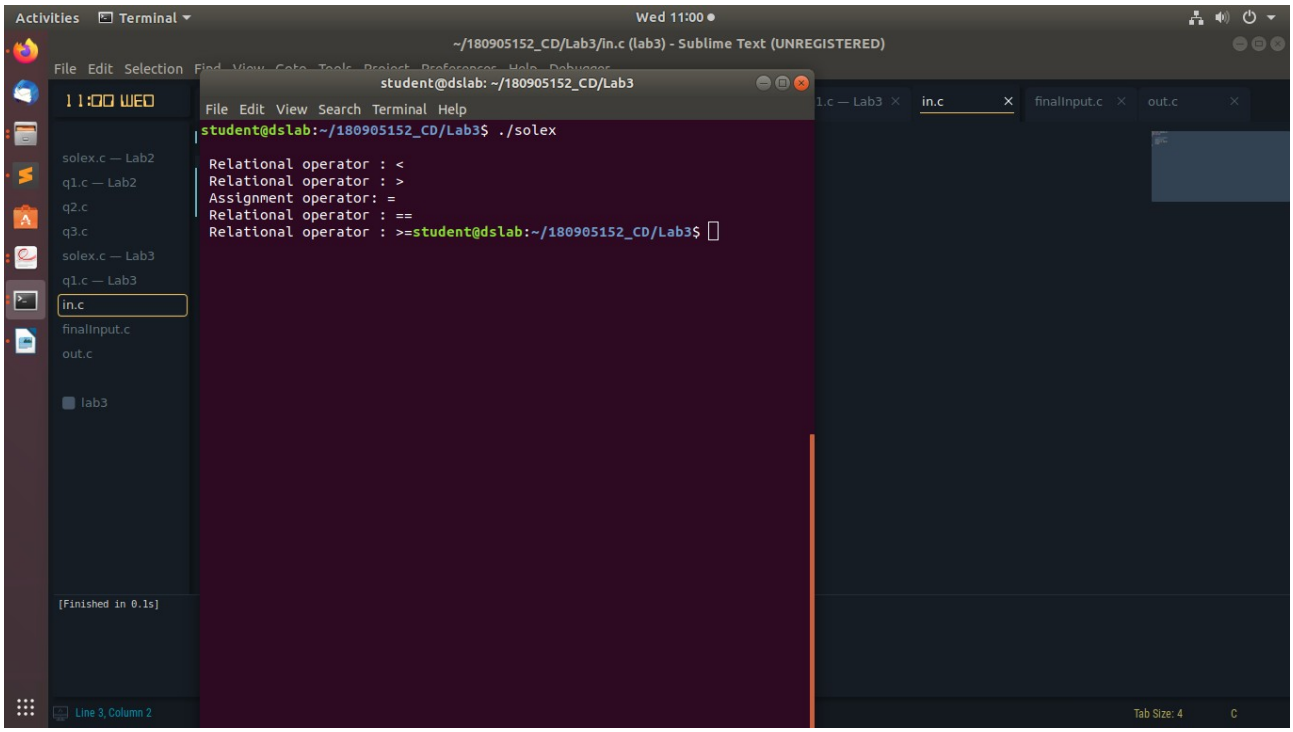
Lab 3

Solved Exercise

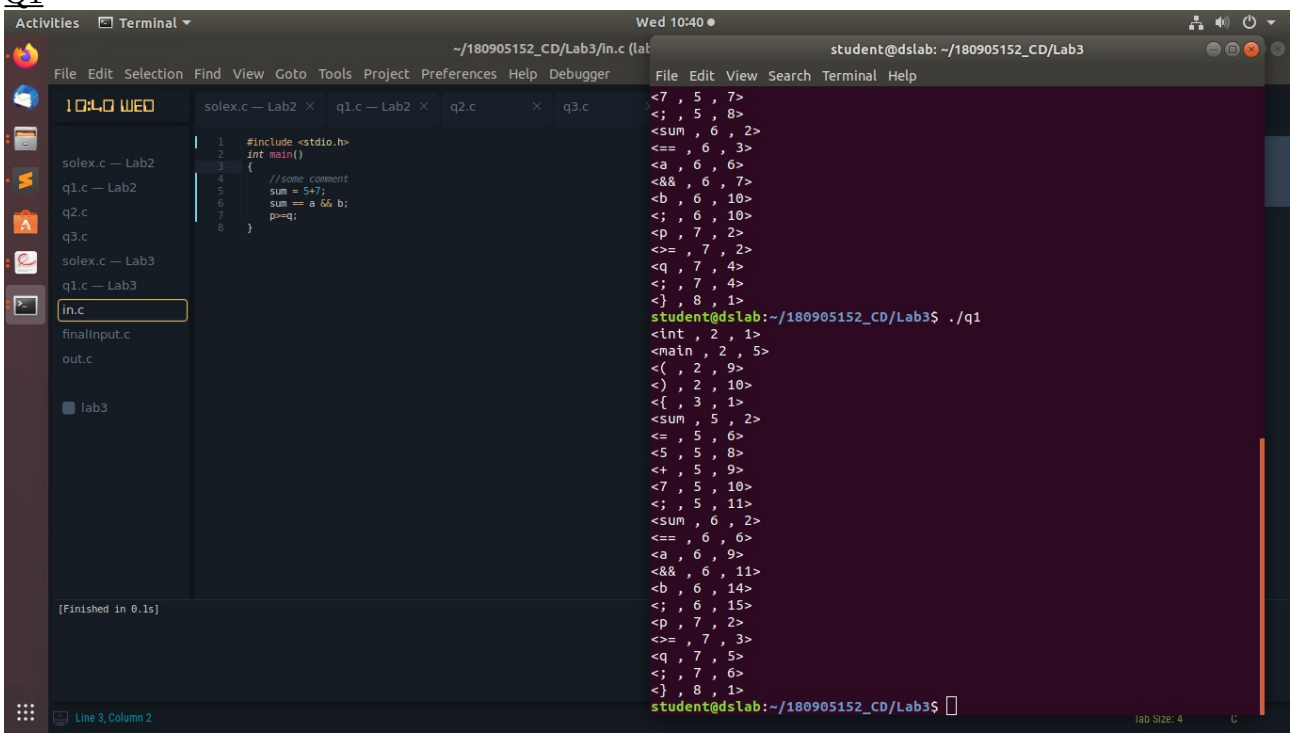


Code

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int main()
{
	char c,buf[10];
	FILE *fp=fopen("in.c","r");
	c = fgetc(fp);
	if (fp == NULL)
	{
		printf("Cannot open file \n");
		exit(0);
	}
	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);
    }
}
```

Q1



Code

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

struct token
{
	char name[20];
	int row,col;
};

char buffer[20];
char keywords[][20] =
{"int","char","float","string","double","if","else","break","continue","for","while","do","main"};
char specialchars[] = {',','.','?',':',';','(',')','{','}'};
char arithmaticsymbols[] = {'+','-','*','/'};
int row = 1,col=0,ca,cb;
FILE *fa,*fb;

struct token getNextToken()
{
	while(ca!=EOF)
	{
		//PreProcessors
		if(ca=='#'){
			while(ca!='\n'){
				col++;
				ca=getc(fa);
			}
		}
		//Comments
		if(ca == '/')
		{
			col++;
			cb = getc(fa);
			if(cb == '/')
			{
				while(ca!='\n')
				{
					col++;
					ca = getc(fa);
				}
			}
			else if(cb == '*')
			{
				do{
					while(ca!='*')
					{
						col++;
						if(ca=='\n'){
							col=1;
```

```
                                        row++;
                                }
                                ca = getc(fa);
                        }
                        ca = getc(fa);
                }while(ca !='/');
        }
}
//Literal
if(ca=='"')
{
        int i=0;
        ca = getc(fa);
        col++;
        while(ca!='"')
        {
                buffer[i++] = ca;
                ca = getc(fa);
        }
        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(fa);
        return t;
}
//blank space
if(ca ==' ')
{
        col++;
        ca = getc(fa);
}
//new line
if(ca == '\n')
{
        row++;
        col=1;
        ca = getc(fa);
}
//Special Character
for(int i=0;i<9;i++)
{
        if(ca==specialchars[i])
        {
```

```
                struct token t;
                t.name[0] = ca;
                t.name[1] = '\0';
                t.row = row;
                t.col = col;
                ca = getc(fa);
                col++;
                return t;
        }
}
//Arithmatic Operators
for(int i=0;i<4;i++)
{
        if(ca == arithmaticsymbols[i])
        {
                struct token t;
                t.name[0] = ca;
                t.name[1] = '\0';
                t.row = row;
                t.col = col;
                ca = getc(fa);
                col++;
                return t;
        }
}
//Assignment or equals op
if(ca == '=')
{
        ca = getc(fa);

        if(ca == '=')
        {
                struct token t;
                t.name[0] = '=';
                t.name[1] = '=';
                t.name[2] = '\0';
                t.row = row;
                t.col = col;
                ca = getc(fa);
                col+=2;
                return t;
        }
        else{
                struct token t;
                t.name[0] = '=';
                t.name[1] = '\0';
                t.row = row;
                t.col = col;
                col++;
                return t;
        }
}
```

```
//RelOps
if(ca == '<')
{
	ca = getc(fa);

	if(ca == '=')
	{
		struct token t;
		t.name[0] = '<';
		t.name[1] = '=';
		t.name[2] = '\0';
		t.row = row;
		t.col = col;
		ca = getc(fa);
		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(fa);

	if(ca == '=')
	{
		struct token t;
		t.name[0] = '>';
		t.name[1] = '=';
		t.name[2] = '\0';
		t.row = row;
		t.col = col;
		ca = getc(fa);
		col+=2;
		return t;
	}
	else{
		struct token t;
		t.name[0] = '>';

		t.name[1] = '\0';
		t.row = row;
		t.col = col;
		col++;
```

```
                return t;
        }
}
//logical ops
if(ca == '&')
{
        ca = getc(fa);

        if(ca == '&')
        {
                struct token t;
                t.name[0] = '&';
                t.name[1] = '&';
                t.name[2] = '\0';
                t.row = row;
                t.col = col;
                ca = getc(fa);
                col+=2;
                return t;
        }
        else{
                struct token t;
                t.name[0] = '&';

                t.name[1] = '\0';
                t.row = row;
                t.col = col;
                col++;
                return t;
        }
}
if(ca == '|')
{
        ca = getc(fa);

        if(ca == '|')
        {
                struct token t;
                t.name[0] = '|';
                t.name[1] = '|';
                t.name[2] = '\0';
                t.row = row;
                t.col = col;
                ca = getc(fa);
                col+=2;
                return t;
        }
        else{
                struct token t;
                t.name[0] = '|';
                t.name[1] = '\0';
                t.row = row;
```

```
                t.col = col;
                col++;
                return t;
        }
}
if(ca == '^')
{
        struct token t;
        t.name[0] = '^';
        t.name[1] = '\0';
        t.row = row;
        t.col = col;
        col++;
        return t;
}
//Numeric
int i=0;
if(isdigit(ca)){
        while(isdigit(ca)){
                buffer[i++] = ca;
                ca = getc(fa);
        }
        buffer[i] = '\0';
        struct token t;
        strcpy(t.name,buffer);
        t.row = row;
        t.col = col;
        col+=strlen(buffer);
        return t;
}
//Keywords
i=0;
while(isalpha(ca)){
        buffer[i++] = ca;
        ca = getc(fa);

}
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(fa);
                col++;
        }
        struct token t;
        t.row = -1;
}

int main()
{

        fa = fopen("in.c","r");
        if(fa == NULL)
        {
                printf("Cannot open file \n");
                return 0;
        }
        ca = getc(fa);
        col = 1;
        while(ca!=EOF)
        {
                struct token t = getNextToken();
                if(t.row == -1)
                        break;
                printf("<%s , %d , %d>\n",t.name,t.row,t.col );
        }
}
```

Name: Shruti Verma
RegNo: 180905152
CSE B -25