Experiment 1

> **Aim**: Design and implement a lexical analyzer for given language using C. The lexical analyzer should ignore redundant spaces, tabs and newlines.

Algorithm

1. Start
2. Read file name of the program: `filename`.
3. Open the file in read mode.
4. If the file does not exist or user do not have the permission to open it, goto step `16`.
5. Read a character from the file into `ch`.
6. While `ch` is not `EOF` (end of file) do
7. If `ch` is '`/`' and next character is '`/`' then skip the line as single line comment.
8. If `ch` is '`/`' and next character is '`*`' keep skipping characters till '`*`' and '`/`' occuring together is encountered: multiline comments.
9. If `ch` is '`#`' skip the line as pre-processor directive.
10. If `ch` is '`"`' keep printing the `ch` till the next '`"`' is encountered. The strings are printed as it is.
11. If `ch` is alphanumeric append `ch` into `token`.
12. Else do
13.
    - If `token` is a keyword print `kwd`.
    - Else if `token` is an identifier print `id`.
    - Else if `token` is a number print the number.
    - Else if `ch` is an operator, print operator token.
    - Else if `ch` is '`,`' or '`;`' print `ch`.
14. Read next character into `ch`
15. Goto step `6`.
16. Close file
17. Stop

Input file

program.c

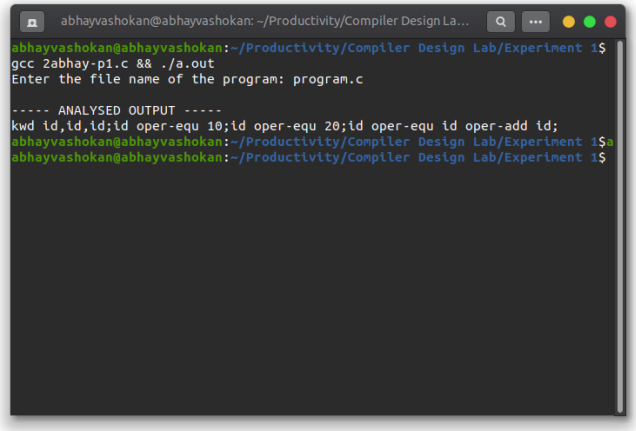
```
// Program to calculate the sum of two numbers

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

void main() {

    /* This is a
```

```
        multiline comment */

    int num1, num2, sum;

    num1 = 10;
    num2 = 20;

    sum = num1 + num2;
}
```

Output

```
Enter the file name of the program: program.c

----- ANALYSED OUTPUT -----
kwd id,id,id;id oper-equ 10;id oper-equ 20;id oper-equ id oper-add id;
```

Screenshot



Readme

1. Compile and run the program using the command `gcc 2abhay-p1.c && a.out`
2. Input the correct path of the file to be analysed.
3. The stream of tokens obtained is displayed in the terminal.