**FORMAN CHRISTIAN COLLEGE (A CHARTERED UNIVERSITY)**

****

**COMP 451 (Compiler Construction)**

**2022 FALL**

**Lab 3 - A**

**Gulraiz Noor Bari (231-525536)**

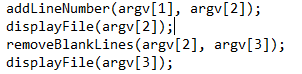
**INTRODUCTION:**

* **stdio.h** (standard input/output)is a header file that contains declarations for functions like, printf, scanf, etc.
* **stdlib.h** (standard library) is a header file that contains declarations of functions that involves memory allocation and process control. **exit(0)** in stdlib.h means to successfully terminate the program. For example, in our program we used **exit(0)** in main() after executing all the user defined functions.
* **string.h** is a header file that contains declarations of functions that are used for working with strings. In our program we have used **strlen** function to get the length of string.
* **main():** The main function serves as the starting point of the program execution in C language. User can pass any number of parameters depending upon the requirements of the program logic or structure. We have passed two parameters:
* **int argc:** Keeps count of the number of command-line arguments entered by the user that also includes the name of the program.
* **char \* argv[]:** Here argv (stores command-line arguments) is an array of pointers that is pointing to the characters of the command-line arguments entered by the user.
* **display():** This method is used to display the file which is stored against the file pointer that is passed as a parameter in the function.
* **addLineNumber():** This method takes two files (input and output) as parameters and reads contents of input file by each character and writes it in the output file with each line preceding with its line number.
* **removeBlankLines():** This method takes two files (input and output), the input file for this function contains the contents of the output file of the **addLineNumber** function which is the file with line numbers preceding each line. This function reads the file by each character and checks for occurrence of two new line **(\n**) characters continuously, if its true then it ignores the empty line and prints the next non-empty line.

**LOGIC/ALGORITHM:**

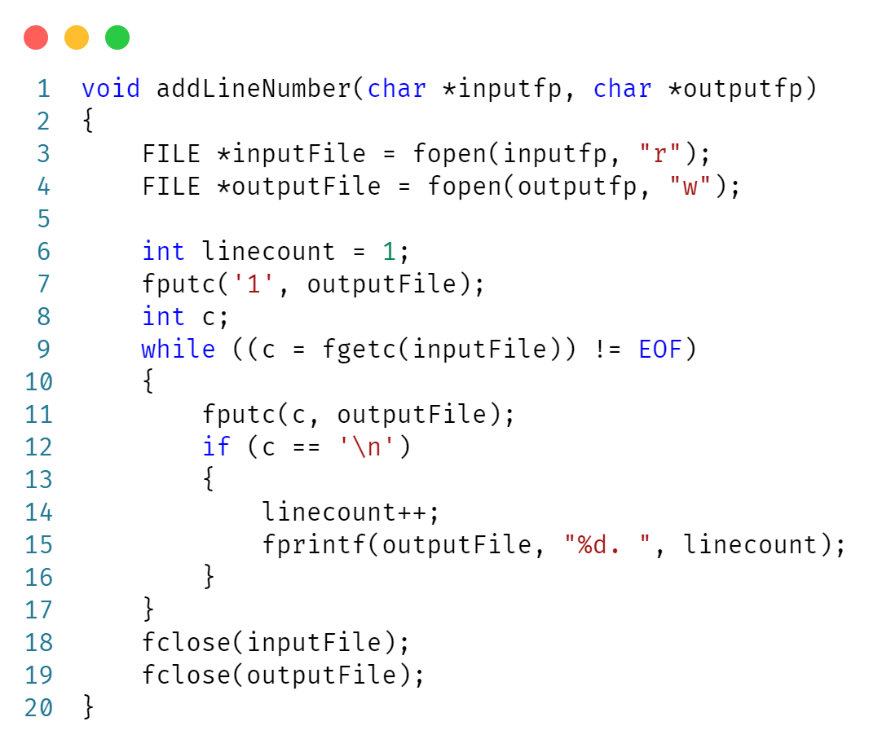
The code is a program that reads a input file by each character and adds a line number preceding each line and removes blank lines from the file and copies the contents in the output file after processing.

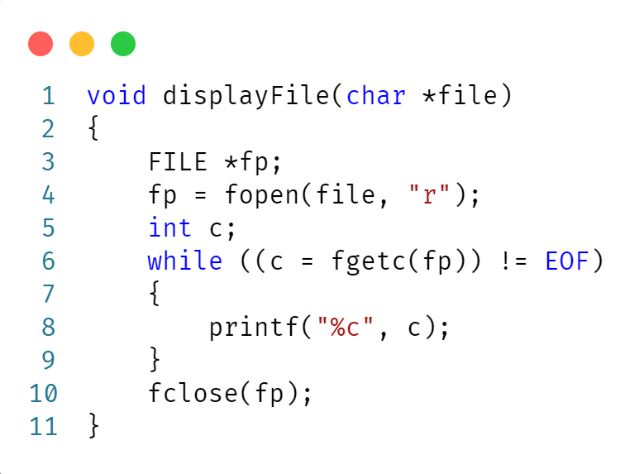
* The program starts by checking whether four arguments were passed by the user in command line, which is done by argc (which has the count of arguments entered in the command-line). If the number of arguments is not four the function is not proceeded and displays a message “Invalid argument”, otherwise if user has entered four arguments; the executable name, input file name, output file in which line numbers should be added and the output file in which blank lines should be removed.
* After checking for argument validity, user defined functions are called in a order specified by the user.

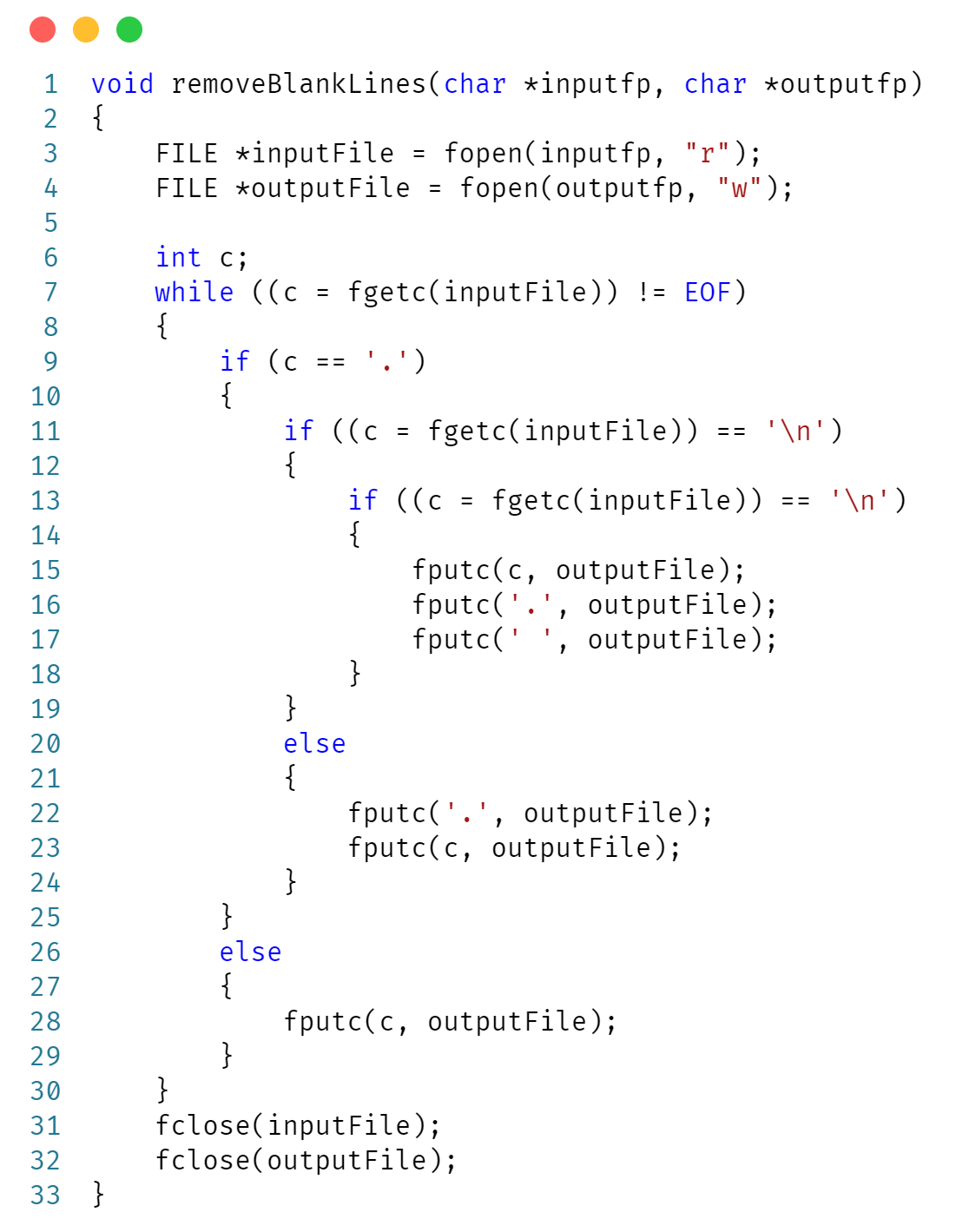


The **addLineNumber** function adds line number to the file given as input by the user (test.txt) and prints the contents of the file in the output file (inFileWithLineNo.txt) then **display** function is called which prints contents of the output file (by reading each character) on the command-line. The **removeBlankLines** function takes the output file (inFileWithLineNo.txt) of the **addLineNumber** function as input and removes the blank lines from the file and prints the new contents in a separate file (inFileWithoutBlanks.txt) as output. **Display** function is called called again which prints the contents of the output file (inFileWithoutBlanks.txt) on the command-line.

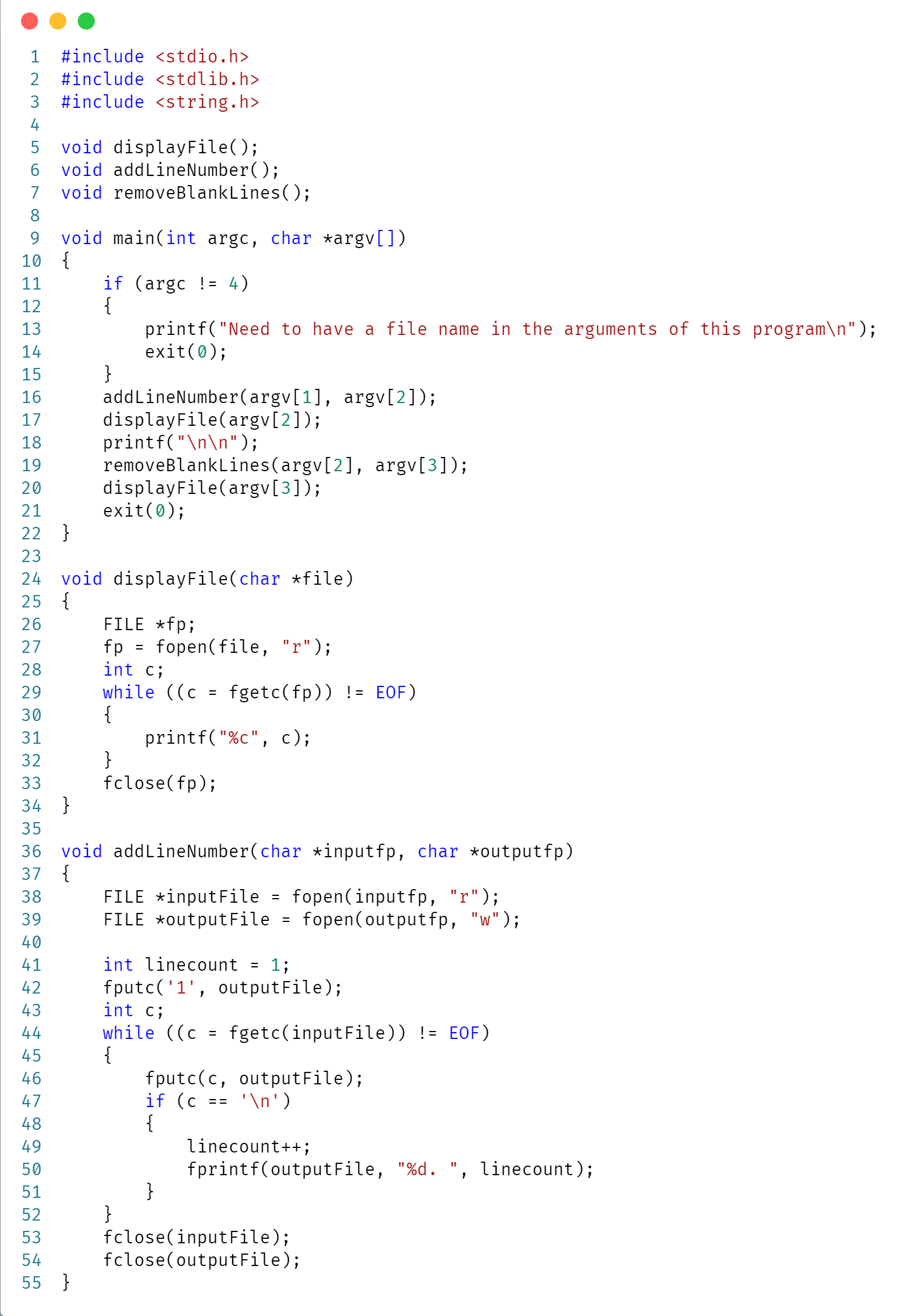
* As following the sequence in main function the program calls the **addLineNumber()** which takes names of two file pointers as input (one is input file and other is output).

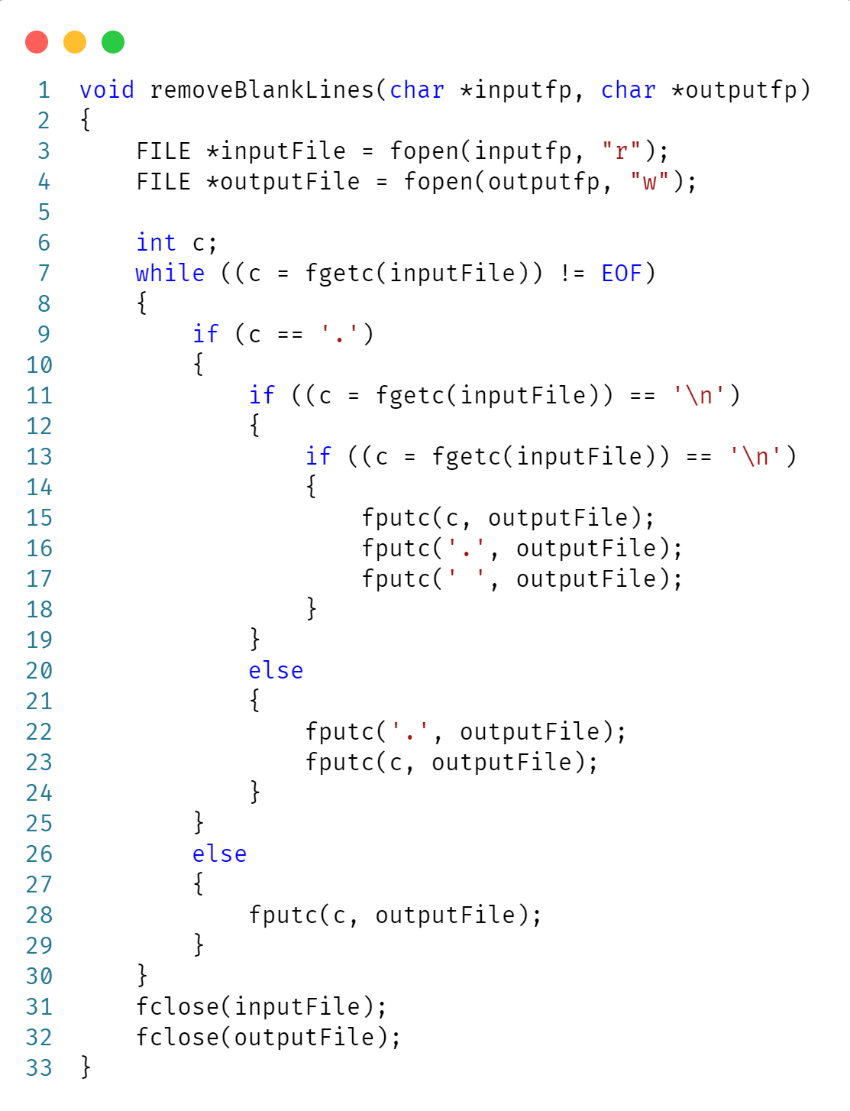


* + Two file pointers (**inputFile** and **outpuFile**) are initialized. Input file pointer points to the file stored in **argv[1]**; as file to be read is second argument user enters in command-line. Output file pointer points to the file stored in **argv[2]**; as file in which the output will be stored is third argument user enters in command-line.
  + A counter is initialized as **linecount**, that keeps count of the number of lines in the text file being read.
  + The **inputFile** is read by each character in a loop using **fgetc** function, which returns the ASCII code of each character and is stored in integer type variable “c”. The loop will run until End of File.
  + In every iteration of the loop/for every character in the file we perform the following checks/operations:
    - After reading one character we use **fputc(c, outputFile)** which prints each character in the output file.
    - If at any point of the program character being read becomes equal to the “**\n**”, the line counter is incremented by 1.
    - **Fprinf** function prints the **linecount** in the output file followed by each line (from the file being read).
    - The loop runs until the character read (“**c**”) becomes equal to **EOF** (End of File) and the contents of the **inputFile** are read by each character and line number is added before each line and the file contents along with line number are copied to **outputFile**.
* After the loop ends both the input and output files are closed using **fclose()**.
* Then **display** function is called in the **main** method which takes name of a file pointer (file which you want to display) as input.
  + The file is read by each character in a loop using **fgetc** function, which returns the ASCII code of each character and is stored in integer type variable “**c**”.
  + The loop prints the contents of the file being read by each character using **“%c”** (character formatter) and stops when End of File is reached.
* Now the program calls the **removeBlankLines** function which takes names of two file pointers as input (one is **inputFile** and other is **outputFile**).

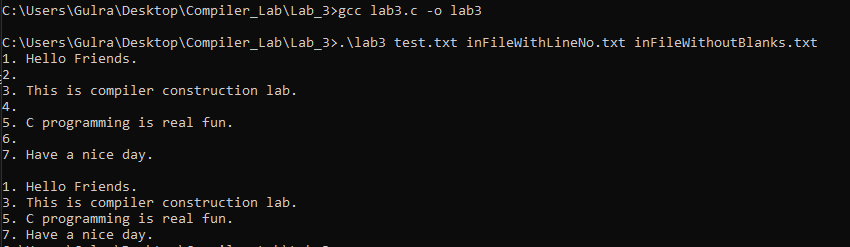


* + Two file pointers (**inputFile** and **outpuFile**) are initialized. Input file pointer points to the file stored in **argv[2]**; which is the file in which **addLineNumber** function writes output. Output file pointer points to the file stored in **argv[3]**; as file in which the output will be stored is forth argument user enters in command-line.
* The **inputFile** is read by each character in a loop using **fgetc** function, which returns the ASCII code of each character and is stored in integer type variable “c”. The loop will run until End of File.
* If at any point of the program character being read is equal to the **(“.”)**; then it checks if there are new line **(‘\n’)** characters occurring twice after fullstop **(“.”)**, if true then it prints the characters of file being read on the next line, in the **outputFile**.
* If there is only one newline character **(‘\n’)** then it means that a newline has started but it is not blank hence it prints the line as it is, in the **outputFile**.
* If **(“.”)** is not found, then it keeps on printing the contents of the **inputFile** in the **outputFile** and ignores the nested if statement
* After the loop ends both the input and output files are closed using **fclose()**.
* Then **display** function is called in the **main** method which takes name of a file pointer (file which you want to display) as input.
  + The file is read by each character in a loop using **fgetc** function, which returns the ASCII code of each character and is stored in integer type variable “**c**”.
  + The loop prints the contents of the file being read by each character using **“%c”** (character formatter) and stops when End of File is reached.
* The program ends with the execution of **exit(0)** statement in **main()**, which refers to successful termination of the program.

**CODE:**



**OUTPUT:**

****