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

****

**COMP 451 (Compiler Construction)**

**2022 FALL**

**Lab 2 - A**

**Muhammad Hamza (231-450349)**

**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. For example, in our program we used **exit(0)** at the end; which means to successfully terminate the program.
* **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.

**LOGIC/ALGORITHM:**

The code is a program that reads a file specified by the user, by each line and displays it on the command-line with each line followed by the number of characters in the specific line and total number of lines in the file are displayed at the end.

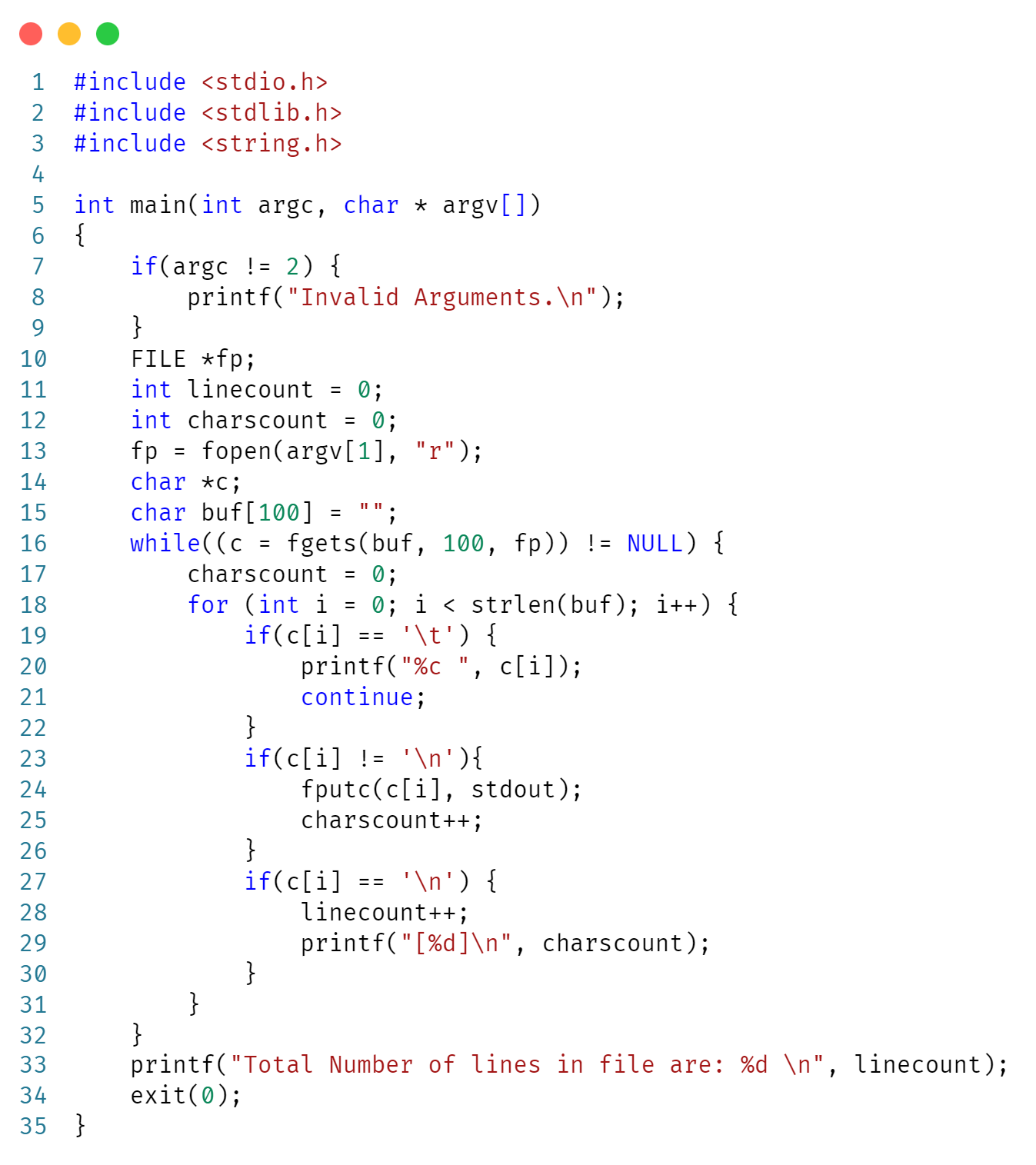
* The program starts by checking whether two arguments were passed by the user in command line, which is done by argc (which has the count of arguments entered in the command-line). If the number of arguments is not two the function is not proceeded and displays a message “Invalid argument”, otherwise if user has entered two arguments; in our case the executable and file name that is to be read, the program proceeds.
* Two counters are initialized:
* **linecount:** Keeps count of the number of lines in the text file.
* **charscount:** Keeps count of the number of characters in the text file.
* Then a file is opened using **fopen** function which takes two parameters; filename and the mode in which the file should be opened, it returns a pointer pointing to the address in the memory where the file is stored. In our case “fp” holds that file pointer that points to the file stored in argv[1], as the file to read is the second argument user enters in the command-line.
* A pointer named “c” of datatype character and a character array with empty strings of size 100 named “buf” are initialized.
* The file is read line by line in a loop using **fgets** function, which takes in three arguments:
  + **char \*str:** Pointer to an array of character where the string read will be copied.
  + **int n:** Maximum number of characters that can be copied.
  + **FILE \*stream:** Pointer to a file that will be read.

**fgets(buf, 100, fp)**

**fgets** function stores first 100 characters of the “fp” file in **buf** array, for each line of every iteration and it returns a string and stores in a pointer of type character declared as “c”. The loop will run until null pointer is returned.

* We run another loop from 0… length of **buf** array; where “i” iterates over the indexes of the string and performs the following checks:
* We check if the index of the string stored in the “c” pointer is not equal to “**\n**” (newline) then we display the file by each character on the command-line and increment **charcount** by 1.
* After reading one character we use fputc(c, stdout) which prints each character on the output; which is the command-line.
* If at character being read is equal to the “**\n**”, the **linecount** is incremented by 1 and **charscount** is displayed on the command-line.
* To remove tab spaces from the program we check whether the “i” index is equal to “**\t**”, if true then keep printing the character being read at the “i” index and continue with the program.
* The loop runs until the character pointer (“c”) becomes equal to NULL and the contents of the file are read by each line and displayed on the command-line and each line followed by the characters count in that line.
* After the loop ends, a line “Total Number of lines in the file” is printed and followed by “%d” (decimal formatter), which replaces **linecount** (stores number of lines in the file).
* The program ends with a exit(0) statement, which refers to successful termination of the program.

**CODE:**

****

**OUTPUT:**