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

****

**COMP 451 (Compiler Construction)**

**2022 FALL**

**Assignment - 1**

**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 (string manipulation).
* **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 removes blank lines, removes comments (both single and multi-line), expands macro’s and prints the final output in output.c file and also in the console.

* 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.
* The input file is opened and read line-by-line (using **fgets**) and printed on the console.
* Then **removeBlankLines()** method is called which takes in two arguments; **input** file and **output** file. A char array named line is initialized, which stores the current line being processed by **fgets()**. Input file is opened and read line-by-line using a while loop and **fgets()**; checks till it encounters NULL, which means the file has been completely read. If the first character of the line is a **‘\n’** (newline character) then it means the line is blank and it skips the line and moves on the next. It keeps on printing the lines one-by-one as it encounters in the loop.
* After this the output file of **removeBlankLines()** method is passed as input to the **removeComments()** method which is used to remove single and multi-line comments in a C-program. It reads the file line-by-line and stores the current line being read in a **char array** named line. It runs a for-loop on each character of the array and checks for:
  + Single line comment (**//**)
  + Start of multi-line comment (**/\***)
  + End of multi-line comment (**\*/**)
  + Multiple occurrences of (**\***) in a multi-line comment

In all the above cases it keeps skipping the characters till it encounters **‘\n’** (newline character) or **‘\0’** (string terminating character) and prints the line by each character in the for-loop.

* The output of above **removeComments()** method is passed through **removeBlankLines()** method; which is called again so we can remove the blank lines that were left after the removal of comments (single**/**multi-line).
* The output of **removeBlankLines()** method is passed through the **macroExpansion()** method which;
  + removes macro lines
  + wherever the macro head is used throughout the program is replaced by macro body
* The input file is opened and read line-by-line. A for-loop is used to iterate over characters of each line being processed (stored in the char array). Following operations are performed in the for-loop:
  + If **‘#’** occurs, then check the next indices till the spaces keep on occurring as; (# define…) is also a valid macro.
  + If **‘d’** occurs, then check the next two indices for **‘e’** and **‘f’**; if **‘def’** it means a macro has been encountered.
  + Now divide the current line into tokens on the basis of spaces between them using the **strtok()** method.

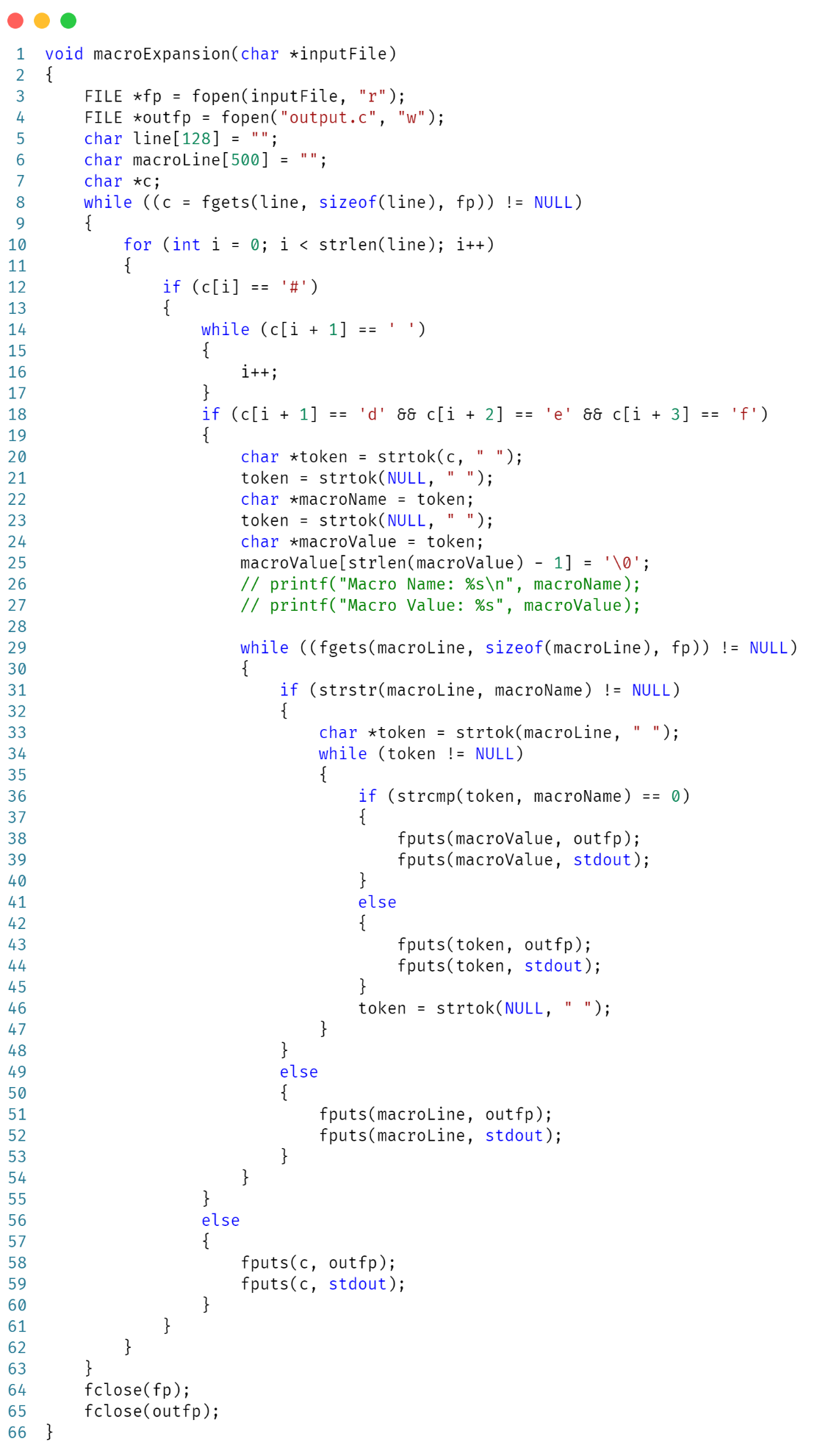
**strtok(NULL, “ ”) :** moves on the next token

We will identify macroName and macroValue using the tokens.

* Proceeding with the loop we the read the file line-by-line using **fgets()** method and store the line being read in **char array** named **macroLine**. We use **strstr()** method to check whether **macroName** exists in **macroLine**,
  + if true we divide the string into tokens using the **strtok(macroLine, “ ”)** method. While token is not NULL we keep on checking for every token generated is equal to macroName; which is done using **strcmp()** method which returns 0 if the strings being compared are equal and if they are equal then the **macroValue** corresponding to that **macroName** is replaced in the program.
  + Else we just print the token in the output file and move to the next token.
* This process repeats till there is no line left to read and all the macros have been expanded and their values have been replaced by macro heads in the program and the program is printed in **“output.c”** file. At the end both the input and output files are closed using **fclose()** method.

Diagram

Description automatically generated with low confidence



**OUTPUT:**

