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

****

**COMP 451 (Compiler Construction)**

**2022 FALL**

**Lab #5**

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

**Muhammad Hamza (231-450349)**

**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 preceding with a line number and it also checks for the number of Macros and displays the macro lines on 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.
* Two counters are initialized:
* **linecount:** Keeps count of the number of lines in the input C file.
* **macroCount:** Keeps count of the number of macros in the input C 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, a character array with empty strings of size 128 named **“buf”** and a char array with empty strings of size 1000 named **“macrosLine”** 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, sizeof(buf), fp)**

**fgets** function stores first 128 (size of buf array) 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.

* The program prints the line number in the while loop; this line number is calculated by iterating over the string (using for-loop) and checking for the newline character (**“\n”**).
* We run a for-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 current index of the string stored in the **“c”** pointer is equal to **“#”**.
* If the above condition is True, then we iterate the immediate next indices and ignore them (by incrementing the iterator **i**) till spaces occur.
* During the above iteration, if at any point the character at the next index is **“d”** (as **#d** means start of a macro) then increment the macroCount by 1.
* Inside the current if-condition we will run a for-loop (from 0….length of buf array) and copy the contents of **buf** array (which will include the contents of macro line) into the **macrosLine** character array, here z iterator will be used for iterating over **macrosLine** array and k for **buf** array. After copying the macroLine we will exit from the for-loop.
* If at character being read (in the buf array) is equal to the “**\n**”, the **linecount** is incremented by 1.
* After repeating the above steps for each character of each line we use **fputc(c[i], stdout)** which prints each character on the output, which is the command-line.
* After the while-loop ends:
  + Contents of the file preceding with line numbers are displayed on the console.
  + “Total Macros found in the file are” is printed and followed by “%d” (decimal formatter), which replaces **macroCount** (stores number of macros in the file).
  + The Macro lines are also displayed below the macro count.
* The file is closed using **fclose()** and program ends with a **exit(0)** statement, which refers to successful termination of the program.

Text

Description automatically generated**CODE:**

**OUTPUT:**

Text

Description automatically generated