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

****

**COMP 451 B**

**SP24**

**Lab 3**

**Hafsah Shahbaz – 251684784**

**Daim Bin Khalid - 251686775**

# **INTRODUCTION**

The program under consideration is a file text processing tool developed in C, designed to address the challenge of cleaning and formatting text files. The primary objectives of this lab revolve around reinforcing fundamental concepts in file handling, character manipulation, and command-line argument parsing within the context of C programming. By tackling this problem, the program aims to provide a practical solution for managing text files by removing excessive spaces while preserving specific formatting rules.

The core problem this program seeks to solve is the inconsistency in text formatting within files. Text files often contain irregular spacing, which can affect readability and processing efficiency. This inconsistency arises due to various factors, including manual editing, data extraction processes, and system interoperability issues. To address this challenge, the program utilizes essential C library functions for file handling and character manipulation. Below are the details of these library functions and their significance within the context of the program.

**1. <stdlib.h>**

The stdio.h library in C provides functions for input and output operations, including file handling. This library is crucial for performing file operations such as opening, reading, and writing files. It enables the program to interact with the user through the console and handle input and output streams efficiently.

1. **fopen():**
   * **Description:** The fopen() function is used to open files and establish a connection between the program and the file.
   * **Usage in the Program:** In our program, fopen() is utilized twice. Firstly, it opens the input file specified by the user for reading. Secondly, it opens an output file named "output.txt" for writing the processed text.
2. **fclose():**
   * **Description:** The fclose() function is employed to close the connection between the program and a file.
   * **Usage in the Program:** After processing the text, fclose() is called to close both the input and output files, ensuring proper resource management and preventing data corruption.
3. **fgetc():**
   * **Description:** The fgetc() function reads a character from the specified file.
   * **Usage in the Program:** In our program, fgetc() is used to read characters from the input file one by one. This enables the program to process the text character by character, applying the necessary formatting rules.
4. **fputc():**
   * **Description:** The fputc() function writes a character to the specified file.
   * **Usage in the Program:** After processing the text, fputc() is utilized to write characters to the output file, generating the cleaned and formatted text. It allows the program to construct the output file character by character, ensuring the desired formatting is applied.
5. **fseek():**
   * **Description:** The fseek() function is used to move the file pointer to a specific position within the file.
   * **Usage in the Program:** In our program, fseek() is employed to navigate within the output file and overwrite specific characters. This functionality is utilized to handle cases where a space needs to be replaced with a semicolon based on formatting rules.

**2. <stdio.h>**

The standard input-output library in C provides functions for reading from and writing to files, as well as console input and output operations. In our program, it's extensively utilized for file handling and console output.

* **printf()**: Used to format and print output to the console. It's employed for displaying informative messages, usage instructions, and file content during program execution.

By leveraging these library functions, the program effectively addresses the text formatting challenge, providing users with a reliable tool for cleaning and formatting text files. Through hands-on experience with file handling and character manipulation, students gain valuable insights into practical programming techniques essential for software development.

# **ALGORITHM and LOGIC**

1. **Argument Check:**
   * **Variables:** argc, argv[]
   * **Description:**
     + Check if the correct number of command-line arguments is provided.
     + If the number of arguments is not equal to 2, display a usage message and exit.
   * **Functions Used:** printf()
2. **Input Validation:**
   * **Variables:** infile
   * **Function Used:** fopen()
   * **Description:**
     + Attempt to open the input file specified by the user.
     + If the file does not exist or cannot be opened, display an error message and exit.
   * **Functions Used:** fopen()
3. **File Opening and Display:**
   * **Variables:** outfile
   * **Functions Used:** fopen()
   * **Description:**
     + Open an output file named "output.txt" for writing.
     + If the file cannot be opened, display an error message and exit.
   * **Functions Used:** fopen()
4. **File Decoding:**
   * **Variables:** ch, next\_ch, space
   * **Functions Used:** fgetc(), fputc(), fseek()
   * **Description:**
     + Read characters from the input file one by one until the end of file (EOF) is reached.
     + While reading each character, check if it's not a newline, tab, or carriage return.
     + If the character is not one of these whitespace characters:
       - If it's a space, check the next character.
         * If the next character is not a semicolon, increment a counter to track consecutive spaces.
         * If it's the first consecutive space, write it to the output file.
         * If the next character is a semicolon, rewind the output file pointer by one position to overwrite the space with the semicolon.
       - If the character is not a space, reset the consecutive space counter and write the character to the output file.
     + If the character is a newline, tab, or carriage return, write it directly to the output file.
   * **Functions Used:** fgetc(), fputc(), fseek()
5. **Program Termination:**
   * **Functions Used:** fclose()
   * **Description:**
     + Close the input and output files before exiting the program.
   * **Functions Used:** fclose()

This algorithm outlines the step-by-step process followed by the program to read input, process the text according to specific rules, and write the processed text to an output file. It involves checking arguments, opening files, reading characters, applying formatting rules, and handling errors to ensure correct execution and proper resource management.

**Code breakdown:**

c

if (argc != 2)

{

printf("Usage: lab3 <filename> \n");

return 1;

}

**Functionality and Logic for Argument Check:**

* This part of the code checks if the correct number of command-line arguments is provided.
* If the argument count (argc) is not equal to 2, it means the user did not provide the input file name along with the program name.
* In such a case, it displays a usage message instructing the user on how to use the program and exits with a non-zero status code.

c

FILE \*infile = fopen(argv[1], "r");

if (infile == NULL)

{

printf("Error opening file %s\n", argv[1]);

return 1;

}

**Functionality and Logic for Input Validation:**

* This part of the code attempts to open the input file specified by the user using fopen() function.
* If the file does not exist or cannot be opened for reading, fopen() returns NULL.
* In such a case, it displays an error message indicating the failure to open the file and exits with a non-zero status code.

c

FILE \*outfile = fopen("output.txt", "w");

if (outfile == NULL)

{

printf("Error opening file output.txt\n");

return 1;

}

**Functionality and Logic for File Opening and Display:**

* This snippet opens an output file named "output.txt" for writing using fopen() function.
* If the file cannot be opened (e.g., due to insufficient permissions or disk space), fopen() returns NULL.
* In such a case, it displays an error message indicating the failure to open the file and exits with a non-zero status code.

c

char ch;

char next\_ch;

int space = 0;

ch = fgetc(infile);

**Explanation:**

* Two variables ch and next\_ch are declared to store the current and next characters read from the input file, respectively.
* An integer variable space is initialized to 0 to keep track of consecutive spaces.
* The fgetc(infile) function reads the first character from the input file infile and assigns it to the variable ch.

while (ch != EOF)

{

next\_ch = fgetc(infile);

**Explanation:**

* Enters a while loop that continues until the end of file (EOF) is reached.
* Inside the loop, the next character from the input file is read and stored in the variable next\_ch

if (ch != '\n' && ch != '\t' && ch != '\r')

{

**Explanation:**

* Checks if the current character ch is not a newline ('\n'), tab ('\t'), or carriage return ('\r').
* If it is not, then if further checks the characters which means it is still processing the current line.

c

if (ch == ' ')

{

if (next\_ch != ';')

{

space += 1;

if (space == 1)

{

fputc(ch, outfile);

}

}

else

{

fseek(outfile, -1, SEEK\_CUR);

}

}

**Explanation:**

* If the current character is a space:
  + And it is not a ‘:’ then simply remove the extra spaces.
  + Checks if the next character (next\_ch) is not a semicolon (;).
  + If it's not a semicolon, increments the space counter to track consecutive spaces.
  + If it's the first consecutive space (space == 1), writes the space to the output file using fputc().
  + If the next character is a semicolon, it means the space should be removed. It moves the output file pointer back one position using fseek() to overwrite the space with the semicolon.

c

else

{

space = 0;

fputc(ch, outfile);

}

**Explanation:**

* If the current character is not a space:
  + Resets the space counter to 0.
  + Writes the character to the output file using fputc().

c

else

{

fputc(ch, outfile);

}

**Explanation:**

* If the current character is a newline, tab, or carriage return:
  + Writes the character to the output file using fputc(). Indication that the current line is read.

c

ch = next\_ch;

**Explanation:**

* Assigns the next character next\_ch to the current character ch to continue processing the next character in the loop.

c

fclose(infile);

fclose(outfile);

return 0;

**Explanation:**

* After processing, this code segment ensures proper cleanup by closing both the input and output files using fclose() function.
* Then, it returns 0 to indicate successful execution of the program.

# **CODE**

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[])

{

if (argc != 2)

{

printf("Usage: lab3 <filename> \n");

return 1;

}

FILE \*infile = fopen(argv[1], "r");

if (infile == NULL)

{

printf("Error opening file %s\n", argv[1]);

return 1;

}

FILE \*outfile = fopen("output.txt", "w");

if (outfile == NULL)

{

printf("Error opening file output.txt\n");

return 1;

}

char ch;

char next\_ch;

int space = 0;

ch = fgetc(infile);

while (ch != EOF)

{

next\_ch = fgetc(infile);

if (ch != '\n' && ch != '\t' && ch != '\r')

{

if (ch == ' ')

{

if (next\_ch != ';')

{

space += 1;

if (space == 1)

{

fputc(ch, outfile);

}

}

else

{

fseek(outfile, -1, SEEK\_CUR);

}

}

else

{

space = 0;

fputc(ch, outfile);

}

}

else

{

fputc(ch, outfile);

}

ch = next\_ch;

}

fclose(infile);

fclose(outfile);

return 0;

# **OUTPUT**

Compile the code: gcc lab3.c -o lab3

**Scenario 1**:

Input.txt:

char myvar = 'a' ;

int i =0;

for (i = 0;i < 3; i++)

{

myvar = myvar \* 2 ;

}

Output.txt

char myvar = 'a';

int i =0;

for (i = 0;i < 3; i++)

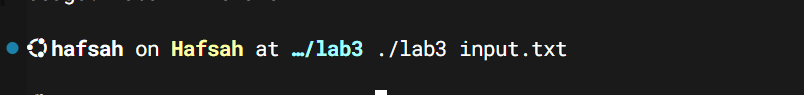
{

myvar = myvar \* 2;

}

A screenshot of a computer

Description automatically generated

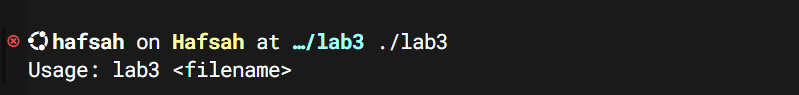


**Comments**:

All unnecessary multiple spaces are removed and only 1 necessary space is kept between each word.

No spaces before semicolons.

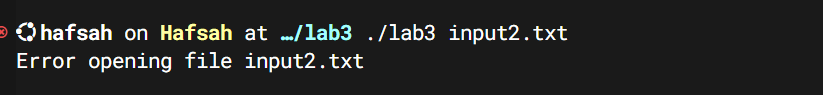
**Scenario 2**:



**Comments**:

Input file is not given as command line argument. So, it displays an error message and returns.

**Scenario 3**:



**Comments**:

Input file which doesn’t exist is given as input. Attempt to open the input file specified by the user.

The file does not exist or cannot be opened, display an error message and return.