Line editor in C/C++

Project Description	**Define Your Project Scope:** you want to create a line editor to read from and writing to files, performing string operations, managing a list of strings, and enabling search and position movement for CRUD operations. **Project Description:** we are going to develop a line editor which should be able to open, write, save files. Open file should be supported from command arguments as defined in step 1 **Create a New C File:** Begin by creating a new C file for your project, e.g., line_editor.c. Include Necessary Header Files: Include standard C libraries like <stdio.h>, <stdlib.h>, <string.h>, etc., for file I/O, memory management, and string operations.
Step 1: Handle Command Line Arguments	Create a program to read commands given at CMD prompt to open a new file. ● **- One Argument (c:/editor):** Create "file.txt" in the current directory with "w+" mode if no additional arguments are provided. ● **- Two Arguments (c:/editor filename):** Open "filename" in "r+" mode if it exists; otherwise, create it in "w+" mode in the current directory. ● **- Three Arguments (c:/editor filename directoryname):** Open "filename" in "directoryname" with "r+" mode if it exists; otherwise, create it in "w+" mode in "directoryname". ● **-more than 3 arguments:** error handling.
Step2:Creating a Buffer	**- Design a buffer** to hold up to 25 lines of text. **- Read the file content** line by line using fgets()/freadline and store each line in the buffer. Allocate memory dynamically for each line. Example to store data in data structure: Suppose we have a text read from opened file then it should store the data linewise: [Line No1:] "This is line 1." [Line No2:] "I love my studies." [Line No3:] "I am in Chitkara University." You should think about a buffer design (How to store data line by line) so that further operations like inserting a line, updating a line, and deleting a line becomes easy.
Step 3: Reading and Writing to the File	● **Reading, storing and printing a file:** Open the file, use fgets() to read lines, and store them in the buffer. Close the file after reading. ● **Writing to file the contents of the buffer:** Open the file in write mode, iterate over the buffer, and use fprintf() to write each line to the file. Close the file after writing.
Step4: Implementing the	● **Search for a specified word in the buffer:** It returns the line number and position of the word if found, return a composite variable with name "cursor"; return -1 if not found.

searchFunction	
Step5	**Implement CRUD Operations on buffer:** ● Write functions to perform CRUD operations (Create(insert), Read, Update, Delete) on lines of text. **Create:** ● Insert a new line at the cursor position. Line should be inserted at the cursor position. If your cursor is at 5th line then a new line will be inserted at 5 position and every other line index will be incremented by one. ● Insert a new word at the cursor position (searched word) and by incrementing the index of further words in this line. **Read (print):** ● Display the contents of the given line. ● Display the whole contents of buffer line wise. **Update** ● Replacing a searched word with another word. ● Insert a new word at cursor position. ● Modify the contents of the current line starting from the current cursor position to the given index or word. **Delete:** ● Remove the current line from the buffer. ● Remove a word (already searched by cursor position) in the given line.