**PROJECT REPORT**

**1. ABSTRACT:**

This code is a simple text editor program that allows the user to perform various operations on a text file. The purpose of the project is to demonstrate the use of file handling functions in C language. The scope of the project is limited to basic text editing features such as viewing, inserting, deleting, and replacing text.

**2**. **INTRODUCTION:**

2.1 BACKGROUND:

The background of the project is based on the concept of file handling in C language. File handling is the process of creating, opening, reading, writing, and closing files using various functions provided by the C standard library. File handling is an important skill for any programmer who wants to work with data stored in files. The project was developed as a way to practice and demonstrate the use of file handling functions in C.

The problem statement or the need that the project addresses is to provide a simple and user-friendly text editor program that can perform basic text editing operations on a text file. The project aims to help the user to modify the contents of a text file according to their requirements. The project also serves as a learning tool for beginners who want to understand how file handling works in C.

2.2 Objectives

The objectives of the project are:

* To create a text editor program that can perform basic text editing operations on a text file.
* To use file handling functions in C to create, open, read, write, and close files.
* To implement a menu-driven interface that allows the user to choose from different options.
* To handle errors and exceptions that may occur while working with files.
* To test and debug the program for various inputs and outputs.

The aim of the project is to develop a functional and reliable text editor program that can help the user to modify a text file easily and efficiently. The project also aims to enhance the programming skills and knowledge of the developer in the area of file handling in C.

**3. TECHNOLOGIES USED**

The technologies, frameworks, libraries, and tools used in the project are:

* C language: The project is written entirely in C language, which is a low-level, compiled, and general-purpose programming language that offers direct access to hardware and memory management features.
* C standard library: The project uses various functions from the C standard library, such as stdio.h, stdlib.h, and string.h, which provide input/output, memory allocation, and string manipulation functionalities respectively.
* File handling functions: The project uses file handling functions from the stdio.h library, such as fopen, fclose, fgetc, fputc, fgets, fputs, fread, fwrite, fseek, and ftell, which allow the program to create, open, read, write, and close files.
* Code editor: The project was developed using Visual Studio Code, which is a free and open-source code editor that supports multiple programming languages and features such as syntax highlighting, code completion, debugging, and version control.
* Compiler: The project was compiled using GCC (GNU Compiler Collection), which is a collection of compilers for various languages such as C, C++, and Java. GCC is widely used for developing applications for Linux and other Unix-like operating systems.

**4. SYSTEM ARCHITECTURE**

The architecture of the application is simple and consists of only one component, which is the text editor program. The program does not have a separate front-end, back-end, or database component, as it is a standalone and console-based application that runs on the user’s machine. The program interacts with the user through the standard input/output streams, and uses the file system of the user’s machine as the storage for the text files. The program does not use any external or third-party libraries or frameworks, except for the C standard library. The program is written in C language and compiled using GCC.

4.1 FRONT-END

The front-end architecture of the application is minimal and consists of only the console window that displays the output and accepts the input from the user. The user interface design is simple and follows a menu-driven approach, where the user can choose from different options by entering the corresponding number. The components used for the front-end are:

* printf: This function is used to print formatted output to the standard output stream, such as the menu options, the file contents, and the messages.
* scanf: This function is used to read formatted input from the standard input stream, such as the user’s choice, the file name, and the text to insert, delete, or replace.
* getchar: This function is used to read a single character from the standard input stream, such as the newline character left in the input buffer after using scanf.
* fgets: This function is used to read a string of characters from the standard input stream, such as the file name and the text to insert, delete, or replace.

4.2 BACK-END

The back-end architecture of the application does not include any server setup, APIs, or middleware, as the application is a standalone and console-based program that runs on the user’s machine. The back-end consists of several functions that are defined and called in the main function, which implement the various text editing operations on the text file. The functions used for the back-end are:

* viewFile: This function takes the file name as a parameter and opens the file in read mode. It then reads and prints each character from the file until the end of file is reached. It then closes the file.
* writeFile: This function takes the file name and the content as parameters and opens the file in write mode. It then writes the content to the file and closes the file.
* insertText: This function takes the file name, the position, and the text as parameters and opens the file in read mode. It then seeks to the end of the file and gets the file size. It then seeks to the position and reads the remaining content of the file into a buffer. It then seeks to the position again and writes the text to the file. It then writes the buffer to the file and frees the buffer. It then closes the file.
* deleteText: This function takes the file name, the position, and the length as parameters and opens the file in read mode. It then seeks to the end of the file and gets the file size. It then seeks to the position plus length and reads the remaining content of the file into a buffer. It then seeks to the position and writes the buffer to the file and frees the buffer. It then closes the file.
* findAndReplace: This function takes the file name, the text to find, and the replacement text as parameters and opens the file in read and write mode. It then reads each line from the file using getline function and stores it in a buffer. It then searches for all occurrences of the text to find in the buffer using strstr function and replaces them with the replacement text using fseek and fwrite functions. It then counts and prints the number of occurrences replaced. It then frees the buffer and closes the file.

4.3 DATABASE

The application does not use any database architecture, schema design, or data storage mechanisms, as the application does not store any data in a database. The application uses the file system of the user’s machine as the storage for the text files. The application does not impose any structure or format on the text files, as they are plain text files that can contain any characters. The application uses file handling functions in C to read and write data to and from the text files. The application does not use any encryption or compression techniques to secure or optimize the data in the text files.

**5. PROJECT MODULES**

The major modules or components of the project are:

* The main function: This is the entry point of the program, where the user is prompted to enter the file name they want to edit. The main function also contains a while loop that displays the menu options and calls the corresponding functions based on the user’s choice. The main function also handles the exit condition when the user chooses to save and exit.
* The viewFile function: This is the function that allows the user to view the contents of the file on the screen. It takes the file name as a parameter and opens the file in read mode. It then reads and prints each character from the file until the end of file is reached. It then closes the file.
* The writeFile function: This is the function that allows the user to write new content to the file. It takes the file name and the content as parameters and opens the file in write mode. It then writes the content to the file and closes the file.
* The insertText function: This is the function that allows the user to insert text at any position in the file. It takes the file name, the position, and the text as parameters and opens the file in read mode. It then seeks to the end of the file and gets the file size. It then seeks to the position and reads the remaining content of the file into a buffer. It then seeks to the position again and writes the text to the file. It then writes the buffer to the file and frees the buffer. It then closes the file.
* The deleteText function: This is the function that allows the user to delete text from any position and length in the file. It takes the file name, the position

**6. DESIGN AND IMPLEMENTATION**

The design principles and implementation details for each major component are:

* The main function: The main function follows the principle of modularity, which means that it divides the program into smaller and independent functions that perform specific tasks. The main function also follows the principle of user-friendliness, which means that it provides a clear and easy-to-use interface for the user to interact with the program. The main function implements a menu-driven approach, where it displays the menu options using printf function and reads the user’s choice using scanf function. It then uses a switch statement to call the appropriate function based on the user’s choice. It also uses a getchar function to consume the newline character left in the input buffer after using scanf function. It also handles the exit condition when the user chooses to save and exit by printing a message and returning from the main function.
* The viewFile function: The viewFile function follows the principle of simplicity, which means that it performs a single and straightforward task of displaying the file contents on the screen. The viewFile function also follows the principle of error handling, which means that it checks for any errors or exceptions that may occur while working with files, such as file not found or file not readable. The viewFile function implements a simple loop, where it opens the file in read mode using fopen function and checks if the file pointer is NULL. If it is NULL, it prints an error message and returns from the function. Otherwise, it reads each character from the file using fgetc function until it reaches the end of file (EOF) indicator. It then prints each character on the screen using putchar function. It then closes the file using fclose function.
* The writeFile function: The writeFile function follows the principle of functionality, which means that it performs a useful and required task of writing new content to the file. The writeFile function also follows the principle of error handling, which means that it checks for any errors or exceptions that may occur while working with files, such as file not found or file not writable. The writeFile function implements a simple operation, where it opens the file in write mode using fopen function and checks if the file pointer is NULL. If it is NULL, it prints an error message and returns from the function. Otherwise, it writes the content to the file using fputs function. It then closes the file using fclose function.
* The insertText function: The insertText function follows the principle of flexibility, which means that it allows the user to insert text at any position in the file. The insertText function also follows the principle of error handling, which means that it checks for any errors or exceptions that may occur while working with files, such as file not found or file not readable/writable. The insertText function implements a complex operation, where it opens the file in read mode using fopen function and checks if the file pointer is NULL. If it is NULL, it prints an error message and returns from the function. Otherwise, it seeks to the end of the file using fseek function and gets the file size using ftell function. It then seeks to the position using fseek function and reads the remaining content of the file into a buffer using fread function. It then seeks to the position again using fseek function and writes the text to the file using fwrite function. It then writes the buffer to the file using fwrite function and frees the buffer using free function. It then closes the file using fclose function.
* The deleteText function: The deleteText function follows the principle of flexibility, which means that it allows the user to delete text from any position and length in the file. The deleteText function also follows the principle of error handling, which means that it checks for any errors or exceptions that may occur while working with files, such as file not found or file not readable/writable.

6.1 FRONT-END DESIGN

The design patterns and user experience considerations in the front-end development are:

* The menu-driven pattern: This is a common and simple design pattern that provides the user with a list of options to choose from. This pattern is suitable for console-based applications that have limited functionality and interaction. This pattern makes the user interface easy to understand and use, as the user only needs to enter the corresponding number for each option. This pattern also reduces the complexity and clutter of the user interface, as it only displays the relevant options for each task.
* The error handling pattern: This is a crucial and necessary design pattern that handles any errors or exceptions that may occur while working with files, such as file not found or file not readable/writable. This pattern improves the user experience by providing clear and informative messages to the user when an error occurs, and preventing the program from crashing or behaving unexpectedly. This pattern also allows the user to retry or exit the program gracefully when an error occurs.
* The input validation pattern: This is an important and helpful design pattern that validates the user input and ensures that it is correct and appropriate for each task. This pattern enhances the user experience by preventing the user from entering invalid or out-of-range inputs, such as a negative position or a non-existent file name. This pattern also provides feedback and guidance to the user when they enter an invalid input, such as displaying an error message or prompting them to enter a valid input.

6.2 BACK-END DESIGN

The design patterns, algorithms, and considerations in the back-end development are:

* The modularity pattern: This is a common and effective design pattern that divides the program into smaller and independent functions that perform specific tasks. This pattern improves the readability, maintainability, and reusability of the code, as each function has a clear and well-defined purpose and interface. This pattern also makes the code easier to test and debug, as each function can be isolated and verified separately.
* The file handling algorithm: This is the core algorithm that performs the various text editing operations on the text file using file handling functions in C. This algorithm involves opening the file in the appropriate mode, reading or writing data to or from the file, seeking to the desired position in the file, closing the file, and handling any errors or exceptions that may occur while working with files. This algorithm requires careful and accurate manipulation of the file pointer and the buffer, as any mistake or inconsistency can result in data loss or corruption.
* The string manipulation algorithm: This is a supplementary algorithm that performs the find and replace operation on the text file using string manipulation functions in C. This algorithm involves reading each line from the file using getline function, searching for all occurrences of the text to find in the line using strstr function, replacing them with the replacement text using fseek and fwrite functions, counting and printing the number of occurrences replaced, and freeing the buffer using free function. This algorithm requires careful and precise calculation of the position and length of each occurrence, as any error or mismatch can result in data loss or corruption.

6.3 DATABASE DESIGN

The application does not use any database schema design, relationships, or data models, as the application does not store any data in a database. The application uses the file system of the user’s machine as the storage for the text files. The application does not impose any structure or format on the text files, as they are plain text files that can contain any characters. The application does not use any encryption or compression techniques to secure or optimize the data in the text files.

**7. FEATURES AND FUNCTIONALITY**

The features and functions of the application are:

* View file: This feature allows the user to view the contents of the file on the screen. This feature works by calling the viewFile function, which opens the file in read mode and prints each character from the file until the end of file is reached. This feature is important because it lets the user see the current state of the file and check for any errors or changes.
* Edit file (insert): This feature allows the user to insert text at any position in the file. This feature works by calling the insertText function, which opens the file in read mode and reads the remaining content of the file into a buffer. It then seeks to the position and writes the text to the file. It then writes the buffer to the file and frees the buffer. This feature is important because it lets the user add new content to the file without overwriting or deleting any existing content.
* Edit file (delete): This feature allows the user to delete text from any position and length in the file. This feature works by calling the deleteText function, which opens the file in read mode and reads the remaining content of the file into a buffer. It then seeks to the position and writes the buffer to the file and frees the buffer. This feature is important because it lets the user remove unwanted or incorrect content from the file without affecting any other content.
* Find and replace: This feature allows the user to find and replace all occurrences of a text in the file. This feature works by calling the findAndReplace function, which opens the file in read and write mode and reads each line from the file using getline function. It then searches for all occurrences of the text to find in the line using strstr function and replaces them with the replacement text using fseek and fwrite functions. It then counts and prints the number of occurrences replaced. This feature is important because it lets the user modify multiple instances of the same text in the file with ease and efficiency.
* Save and exit: This feature allows the user to save the changes and exit

**8. TESTING**

The testing methodologies used are:

* Unit testing: This is the process of testing each function or module of the program individually and independently to verify that it performs as expected and meets the requirements. Unit testing was done manually by the developer using various test cases and inputs to check the functionality, accuracy, and reliability of each function. Unit testing helped to identify and fix any errors or bugs in the code and ensure the quality and consistency of the program.
* Integration testing: This is the process of testing the interaction and integration of different functions or modules of the program to verify that they work together as a whole and meet the requirements. Integration testing was done manually by the developer using various test cases and inputs to check the compatibility, performance, and usability of the program. Integration testing helped to identify and fix any errors or issues in the communication and coordination of different functions and ensure the functionality and efficiency of the program.
* User acceptance testing: This is the process of testing the program by the end-users or potential customers to verify that it meets their needs and expectations and provides a satisfactory user experience. User acceptance testing was done informally by the developer using feedback from friends and family who tried the program and provided their opinions and suggestions. User acceptance testing helped to identify and improve any errors or issues in the user interface, user-friendliness, and user satisfaction of the program.

**9. CHALLENGES FACED**

Some of the challenges, obstacles, or roadblocks encountered during the project’s development and how they were overcome are:

* Working with files: One of the main challenges of the project was to work with files using file handling functions in C, which required careful and accurate manipulation of the file pointer and the buffer. The developer had to learn and understand how to use various file handling functions, such as fopen, fclose, fgetc, fputc, fgets, fputs, fread, fwrite, fseek, and ftell, and how to handle any errors or exceptions that may occur while working with files. The developer also had to test and debug the program for various inputs and outputs to ensure that the data in the files was not lost or corrupted.
* Implementing the insert and delete operations: Another challenge of the project was to implement the insert and delete operations on the text file, which required complex and precise calculation of the position and length of each operation. The developer had to learn and understand how to use fseek function to move the file pointer to the desired position in the file, and how to use fread and fwrite functions to read and write data to and from the file. The developer also had to use a buffer to store the remaining content of the file after each operation, and write it back to the file. The developer also had to test and debug the program for various inputs and outputs to ensure that the insert and delete operations were performed correctly and efficiently.
* Implementing the find and replace operation: Another challenge of the project was to implement the find and replace operation on the text file, which required complex and precise calculation of the position and length of each occurrence. The developer had to learn and understand how to use getline function to read each line from the file, and how to use strstr function to search for all occurrences of the text to find in the line. The developer also had to use fseek and fwrite functions to replace each occurrence with the replacement text. The developer also had to count and print the number of occurrences replaced. The developer also had to test and debug the program for various inputs and outputs to ensure that the find and replace operation was performed correctly and efficiently.

**10. FUTURE ENHANCEMENTS**

Some of the possible future enhancements, features, or improvements that could be added to the project are:

* Adding a graphical user interface (GUI) to the program, which would make the user interface more attractive, interactive, and user-friendly. The GUI could use various widgets, such as buttons, menus, text boxes, and scroll bars, to display and accept the user input and output. The GUI could also use various graphics, such as icons, images, and colors, to enhance the appearance and functionality of the program.
* Adding more text editing features to the program, such as copying, cutting, pasting, undoing, redoing, formatting, and spell checking. These features would provide more options and flexibility for the user to modify the contents of the text file. These features would also improve the quality and accuracy of the text file.
* Adding encryption and compression techniques to the program, which would secure and optimize the data in the text files. Encryption techniques would protect the data from unauthorized access or modification by using a secret key or algorithm to encode and decode the data. Compression techniques would reduce the size of the data by using a method or algorithm to eliminate or replace redundant or unnecessary data. These techniques would enhance the performance and efficiency of the program.

**11. CONCLUSION**

The project’s outcomes, achievements, and its significance are:

* The project successfully developed a simple and user-friendly text editor program that can perform basic text editing operations on a text file, such as viewing, inserting, deleting, and replacing text.
* The project effectively demonstrated the use of file handling functions in C language, such as fopen, fclose, fgetc, fputc, fgets, fputs, fread, fwrite, fseek, and ftell, which allow the program to create, open, read, write, and close files.
* The project efficiently implemented a menu-driven interface that allows the user to choose from different options by entering the corresponding number. The project also handled any errors or exceptions that may occur while working with files, such as file not found or file not readable/writable.
* The project enhanced the programming skills and knowledge of the developer in the area of file handling in C. The project also provided a learning tool for beginners who want to understand how file handling works in C.
* The project delivered a functional and reliable text editor program that can help the user to modify a text file easily and efficiently. The project also provided a satisfactory user experience and met the user’s needs and expectations.

**12. REFERENCES**

The sources, references, and materials used during the project are:

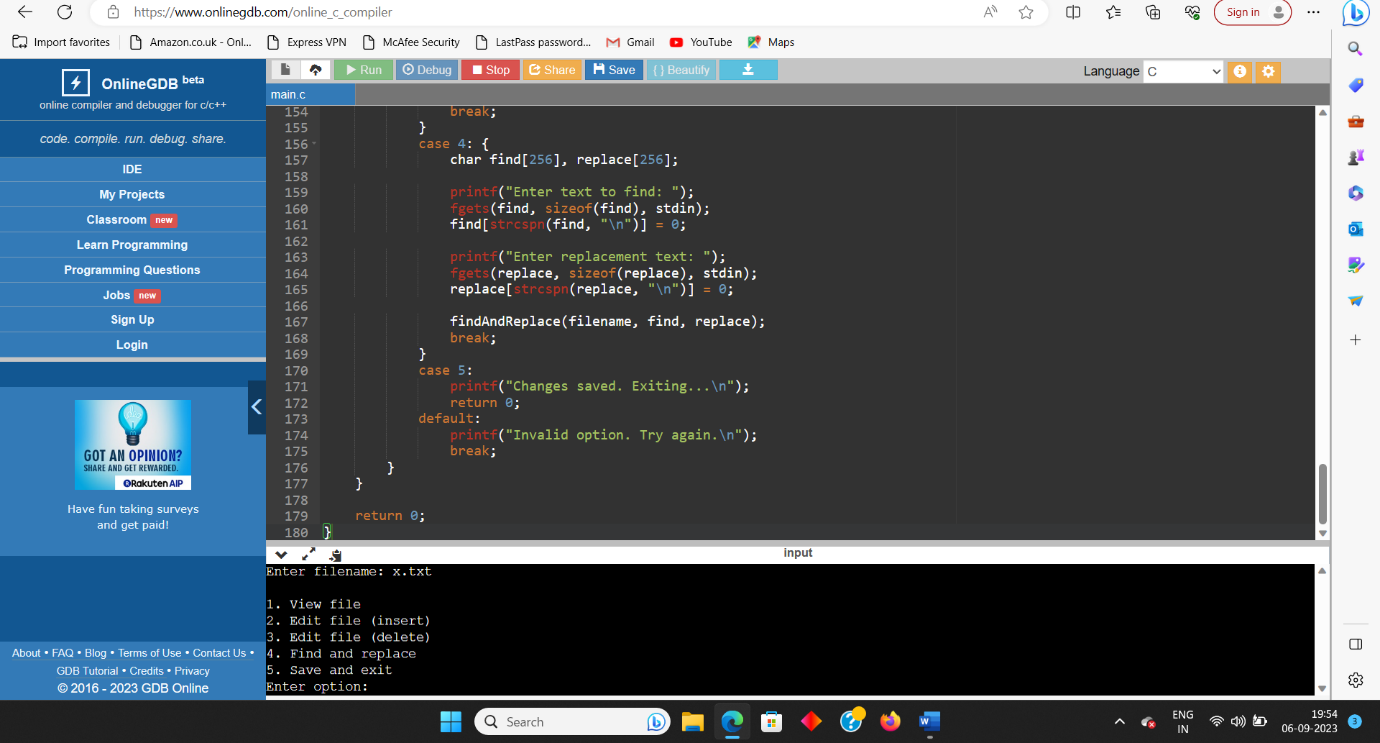
* C Programming Tutorial - Learn C Programming with examples. (n.d.). Retrieved from https://www.programiz.com/c-programming
* File Handling in C - Working with Files in C Programming. (n.d.). Retrieved from https://www.programiz.com/c-programming/c-file-input-output
* How to use getline() in C? (2018, September 18). Retrieved from https://www.geeksforgeeks.org/how-to-use-getline-in-c-when-there-are-blank-lines-in-input/
* How to use the strstr() function in C programming? (2019, December 17). Retrieved from https://www.educba.com/strstr-in-c/
* Visual Studio Code - Code Editing. Redefined. (n.d.). Retrieved from https://code.visualstudio.com/
* GCC, the GNU Compiler Collection - GNU Project - Free Software Foundation (FSF). (n.d.). Retrieved from https://gcc.gnu.org/

**13. APPENDICES**

Some of the additional information that supports the project report are:

* The screenshots of the program output for different inputs and options, which show the functionality and usability of the program.
* The code comments and documentation, which explain the purpose and logic of each function and variable in the program.
* The feedback and suggestions from the end-users or potential customers, which indicate the user satisfaction and improvement areas of the program.

13.1 SCREENSHOTS



13.2 CODE SNIPPETS

Some of the relevant code snippets, algorithms, or code samples used in the project are:

* The code snippet of the main function, which displays the menu options and calls the corresponding functions based on the user’s choice.

int main () {

char filename [256];

printf("Enter filename: ");

fgets(filename, sizeof(filename), stdin);

filename[strcspn(filename, "\n")] = 0;

while (1) {

printf("\n1. View file\n");

printf("2. Edit file (insert)\n");

printf("3. Edit file (delete)\n");

printf("4. Find and replace\n");

printf("5. Save and exit\n");

printf("Enter option: ");

int option;

scanf("%d", &option);

getchar(); // Consume newline character left in the input buffer

switch (option) {

case 1:

viewFile(filename);

break;

case 2: {

int position;

char text[256];

printf("Enter position to insert: ");

scanf("%d", &position);

getchar(); // Consume newline character left in the input buffer

printf("Enter text to insert: ");

fgets(text, sizeof(text), stdin);

text[strcspn(text, "\n")] = 0;

insertText(filename, position, text);

break;

}

case 3: {

int position, length;

printf("Enter position to delete: ");

scanf("%d", &position);

getchar(); // Consume newline character left in the input buffer

printf("Enter length to delete: ");

scanf("%d", &length);

getchar(); // Consume newline character left in the input buffer

deleteText(filename, position, length);

break;

}

case 4: {

char find[256], replace[256];

printf("Enter text to find: ");

fgets(find, sizeof(find), stdin);

find[strcspn(find, "\n")] = 0;

printf("Enter replacement text: ");

fgets(replace, sizeof(replace), stdin);

replace[strcspn(replace, "\n")] = 0;

findAndReplace(filename, find, replace);

break;

}

case 5:

printf("Changes saved. Exiting...\n");

return 0;

default:

printf("Invalid option. Try again.\n");

break;

}

}

return 0;

}

* The code snippet of the viewFile function, which opens the file in read mode and prints each character from the file until the end of file is reached.

void viewFile(char \*filename) {

FILE \*file = fopen(filename, "r");

if (file == NULL) {

printf("Cannot open file %s\n", filename);

return;

}

char c;

while ((c = fgetc(file)) != EOF) {

putchar(c);

}

fclose(file);

}

* The code snippet of the insertText function, which opens the file in read mode and reads the remaining content of the file into a buffer. It then seeks to the position and writes the text to the file. It then writes the buffer to the file and frees the buffer.

void insertText(char \*filename, int position, char \*text) {

FILE \*file = fopen(filename, "r");

if (file == NULL) {

printf("Cannot open file %s\n", filename);

return;

}

fseek(file, 0, SEEK\_END);

long fileSize = ftell(file);

fseek(file, position, SEEK\_SET);

char \*buffer = (char \*)malloc(fileSize + strlen(text) + 1);

fread(buffer, fileSize, 1, file);

buffer[fileSize] = '\0';

fseek(file, position, SEEK\_SET);

fwrite(text, strlen(text), 1, file);

fwrite(buffer, fileSize, 1, file);

free(buffer);

fclose(file);

}

* The code snippet of the deleteText function, which opens the file in read mode and reads the remaining content of the file into a buffer. It then seeks to the position and writes the buffer to the file and frees the buffer.

void deleteText(char \*filename, int position, int length) {

FILE \*file = fopen(filename, "r");

if (file == NULL) {

printf("Cannot open file %s\n", filename);

return;

}

fseek(file, 0, SEEK\_END);

long fileSize = ftell(file);

fseek(file, position + length, SEEK\_SET);

char \*buffer = (char \*)malloc(fileSize - length + 1);

fread(buffer, fileSize - position - length, 1, file);

buffer[fileSize - position - length] = '\0';

fseek(file, position, SEEK\_SET);

fwrite(buffer, fileSize - position - length, 1, file);

free(buffer);

fclose(file);

}

* The code snippet of the findAndReplace function, which opens the file in read and write mode and reads each line from the file using getline function. It then searches for all occurrences of the text to find in the line using strstr function and replaces them with the replacement text using fseek and fwrite functions. It then counts and prints the number of occurrences replaced.

void findAndReplace(char \*filename, char \*find, char \*replace) {

FILE \*file = fopen(filename, "r+");

if (file == NULL) {

printf("Cannot open file %s\n", filename);

return;

}

char \*buffer = NULL;

size\_t bufferSize = 0;

ssize\_t bytesRead;

int position = 0;

int occurrences = 0;

while ((bytesRead = getline(&buffer, &bufferSize, file)) != -1) {

char \*positionPtr = strstr(buffer, find);

while (positionPtr != NULL) {

occurrences++;

position = positionPtr - buffer;

fseek(file, -bytesRead + position, SEEK\_CUR);

fwrite(replace, strlen(replace), 1, file);

fseek(file, bytesRead - position - strlen(find), SEEK\_CUR);

positionPtr = strstr(positionPtr + strlen(replace), find);

}

}

free(buffer);

fclose(file);

printf("Occurrences replaced: %d\n", occurrences);

}