Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Computer Programming

Laboratory work #4

Character and string operations

Author:
Alexandru Rudoi
std. gr. FAF-231

Verified:
Alexandru Furdui

Chișinău 2023

Theory Background

Buffer Management

Buffer: In computing, a buffer is a temporary storage area that holds data as it's being transferred from one place to another. In the context of this program, a buffer is used to store the text entered by the user.

Dynamic Memory Allocation

- **malloc()**: This function is used to dynamically allocate memory on the heap during program execution. It allocates a block of memory of a specified size and returns a pointer to the first byte of the block. In the program, malloc() is used to allocate memory for the text buffer.

- **realloc()**: When the buffer size needs to be increased (e.g., when the entered text exceeds the current buffer size), realloc() is used to resize the allocated memory block.

File I/O (Input/Output)

- **File Pointer (FILE*)**: A file pointer is a variable that points to a FILE structure, which is used to manage input and output streams to files. In the program, FILE* is used to open, read, and write files.

- **fopen()**: This function is used to open files. It takes the filename and a mode (e.g., ”r” for reading and ”w” for writing) as arguments.

- **fclose()**: This function is used to close files that have been opened with fopen(). It's essential to close files after using them to free up system resources.

- **fprintf()**: This function is used to write formatted data to a file. In the program, it's used to save the buffer's content to a file.

- **fgets()**: This function is used to read a line from a file. It reads characters from the file until a newline character (”) is encountered.

Character and String Operations

- **strlen()**: This function calculates the length of a string, which is the number of characters in the string (excluding the null-terminator).

- **strcpy()**: This function copies the characters from one string to another, effectively appending text to the buffer.

- **strcspn()**: It's used to remove the newline character (") from input strings by finding the position of the newline character and replacing it with ".

Exception Handling

- **setjmp() and longjmp()**: These functions are used for basic exception handling. setjmp() saves the program's current state, and longjmp() jumps back to this state when an exception is encountered. In the program, these are used to implement a rudimentary TRY-CATCH mechanism for file I/O errors.

ANSI Escape Codes for Colors

The program uses ANSI escape codes to control the color of text in the console. These codes are not specific to C but are used to add color and formatting to text displayed in the console. For example, `RED_TEXT` sets text to red, and `RESET_COLOR` resets it to the default color.

The Task

Implement a simple text editor program in C that allows users to enter, edit, search, and manipulate text within a buffer. The program should provide a menu-based interface with the following functionalities:

- Enter Text: it will append the entered text to the existing text
- Search for Word: it will display the number of occurrences
- Replace Word
- Delete: it should delete the entire buffer
- Save to File(optional)
- Load from File(optional)
- Exit

Task Requirements:

• Use appropriate data structures (e.g. arrays or linked lists) to manage the text in the buffer, handle search and replace operations, and store search results.

• Ensure proper memory management and error handling, including file I/O errors(if you will implement files functionality) and buffer overflow protection.

• Provide clear and user-friendly prompts and messages to guide users through the menu options and operations. If you feel like adding more functionalities, feel free to do so, it will only benefit you:)

Technical implementation

Listing of the program: Github

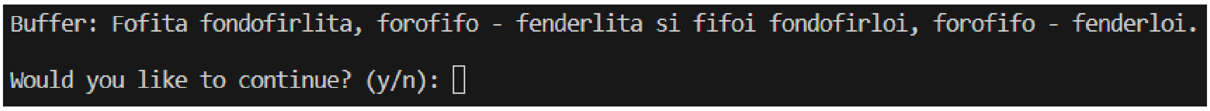
Pseudo-code:

```
Function enterText (text: *char) -> int
    max_size = 10
    current_size = 0
    Allocate memory for text with max_size

    if allocation fails, return 1

    loop while true
        c = read character from input

        if c is EOF or newline character
            Null-terminate text and break from loop
        append c to text
        increment current_size

        if current_size >= max_size
            max_size *= 2
            Resize text to max_size

            if reallocation fails, return 1

    return 0

Function searchWord(text: *char, word: *char) -> int
    if text or word is NULL, return -1
    text_length = length of text
    word_length = length of word
    count = 0

    loop for i from 0 to (text_length - word_length)
        match = 1

        loop for j from 0 to (word_length - 1)
            if text[i + j] is not equal to word[j]
                match = 0
                break

        if match is 1
            if (i is 0 or previous character is not alphanumeric) and
               (i + word_length is equal to text_length or next character is not alphanumeric)
                increment count
                skip word in the text

    return count

Function replaceWord(text: *char, word: *char, replacement: *char) -> int
    if text is NULL, return 1
    text_length = length of text
    word_length = length of word
    replacement_length = length of replacement
    Allocate memory for result with text_length + 1
    if allocation fails, return 1
    i = 0
    j = 0

    loop while i is less than text_length
        match = 1

        loop for k from 0 to (word_length - 1)
            if text[i + k] is not equal to word[k]
                match = 0
                break

        if match is 1
            copy replacement into result
            increment j by replacement_length
            increment i by word_length
        else
            copy text[i] into result[j]
            increment j
            increment i

    Null-terminate result

    free old text
    update text with result

    return 0

Function deleteBuffer(text: *char) -> int
    if text is NULL, return 1
    free text
    update text to NULL
    return 0
```

Results

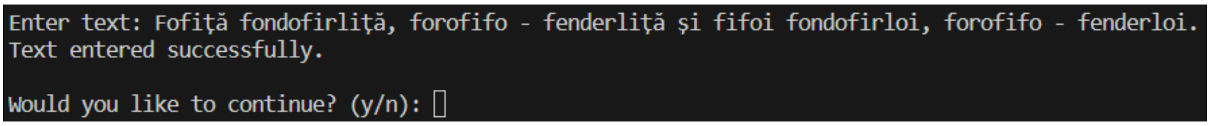
This program it presents a menu to the user, allowing various operations on a text buffer:

1. Print Buffer: Displays the contents of the text buffer.
2. Enter Text: Allows the user to input text, and this text is appended to the existing buffer.
3. Search for a Word: Prompts the user to enter a word to search within the text buffer and then displays the count of occurrences of that word.
4. Replace a Word: Enables the user to replace all occurrences of a word in the text buffer with another word.
5. Delete Buffer: Clears the entire text buffer.
6. Save to File: Saves the content of the buffer to a file named ”text.txt”.
7. Load from File: Loads the content of the ”text.txt” file into the buffer.
8. Exit: Exits the program. In the next photos, there are some examples:



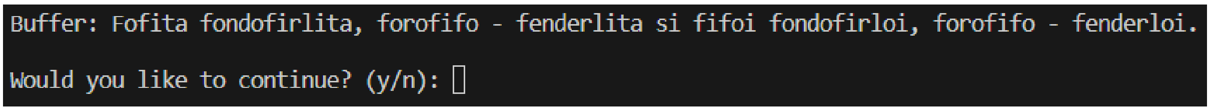
```
Buffer: Fofita fondofirlita, forofifo - fenderlita si fifoi fondofirloi, forofifo - fenderloi.

Would you like to continue? (y/n):
```

Figure 1



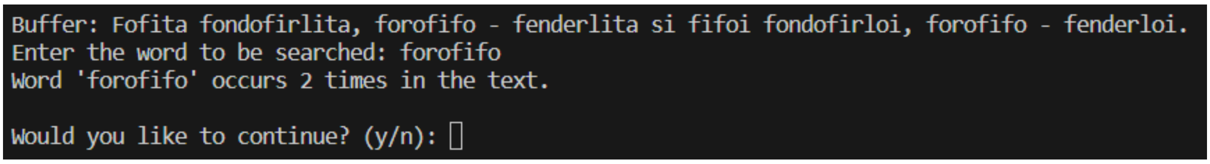
```
Enter text: Fofiță fondofirliță, forofifo - fenderliță și fifoi fondofirloi, forofifo - fenderloi.
Text entered successfully.

Would you like to continue? (y/n):
```

Figure 2



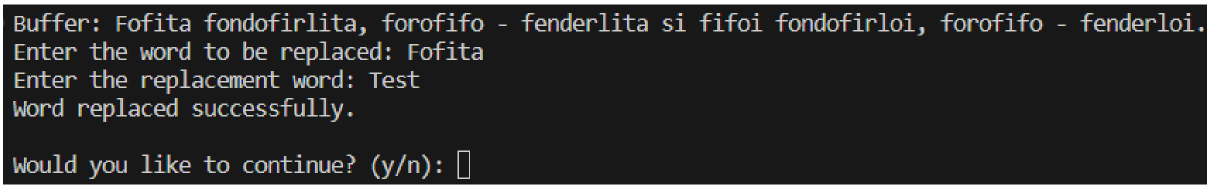
```
Buffer: Fofita fondofirlita, forofifo - fenderlita si fifoi fondofirloi, forofifo - fenderloi.

Would you like to continue? (y/n):
```

Figure 3



```
Buffer: Fofita fondofirlita, forofifo - fenderlita si fifoi fondofirloi, forofifo - fenderloi.
Enter the word to be searched: forofifo
Word 'forofifo' occurs 2 times in the text.

Would you like to continue? (y/n):
```

Figure 4



```
Buffer: Fofita fondofirlita, forofifo - fenderlita si fifoi fondofirloi, forofifo - fenderloi.
Enter the word to be replaced: Fofita
Enter the replacement word: Test
Word replaced successfully.

Would you like to continue? (y/n):
```

Figure 5

Conclusion

In summary, this laboratory work has been instrumental in equipping me with a diverse skill set that transcends C and applies to a broad spectrum of software development challenges. It has not only deepened my understanding of C programming but also enhanced my knowledge of memory management, file handling, string manipulation, and exception handling. These skills will undoubtedly prove invaluable in my journey as a future software engineer, enabling me to tackle complex data structures and algorithmic challenges with confidence.

Bibliography

1. The overview of Computer Programming course lessons for students (lecturer: associate professor M. Kulev). Chisinau, UTM, FCIM, 2023.

2. https://www.programiz.com/c-programming/c-strings

3. https://www.tutorialspoint.com/cprogramming/c$_{s}trings.htm$

4. https://www.geeksforgeeks.org/string-functions-in-c/

5. https://beginnersbook.com/2014/01/c-strings-string-functions/