

Text-Terminal: A UTF-8 Text Editor for the Linux Shell

Project Report
Operating Systems Lecture Spring Semester 2025

University of Basel
Faculty of Science
Department of Mathematics and Computer Science

Heinrich Ahrend
Yorik Metzger
Jorn Riedel
Shura V. Ruben

June DD, 2025



Contents

1	Introduction	1
2	Background	1
3	Methodology	1
4	Results	2
5	Discussion	3
6	Conclusion	3
7	Lessons Learned	3
	References	3
	Appendix A: Plots	4
	Appendix B: Materials	4
	B.1 Software Requirements	4
	Appendix C: Declaration of Independent Authorship	5

1 Introduction

Text editors are an essential tool that is quite often taken for granted since pretty much every single operating system has one included by default. Despite this, quite a lot of software engineering is required to develop a software product that is able to full fill the requirements posed by the tasks we need and wish to perform in our text editors. This is especially the case when features like word and line count should work well with very large files.

Since these aspects intrigued us and we wanted to try and develop our own solution from the ground up, we choose to develop a full text editor as our OS course project. More specifically we have opted to try and develop a plain text (file) editor, with good handling of large (even very large files), since this is one of the major weak points we identified with VS Code and other text editors. Additionally:

- it should support UTF-8 encoding since it is essential when writing a text in German and French.
- it should be compatible with the three most common line break standards: `\n` (Linux), `\r\n` (Windows), `\r` (Mac).
- the user interface should be a bit easier to use then pure keyboard text editors (e.g. vi/vim); it should allow to use the mouse for most important operations.
- and finally the user interface should be responsive even when operating on large files.

2 Background

ncursesw version 6 [1] is the library that we chose to build our text-based UI because it has good terminal support. Specifically we chose the wide characters version of ncurses so that it can support Unicode and international character sets (standard ncurses only has ASCII support).

We also used xclip [2] for our copy/paste functionality. It allows accessing the clipboard in X11, which is the Linux desktop environment. It bridges the terminal and GUI clipboard.

3 Methodology

After deciding on our project topic we fixed our software requirements in more concrete form. Having done so we identified that it would be most beneficial to divide our future code into front end and backed with the main interface being between graphical user interface (GUI) and text data structure.

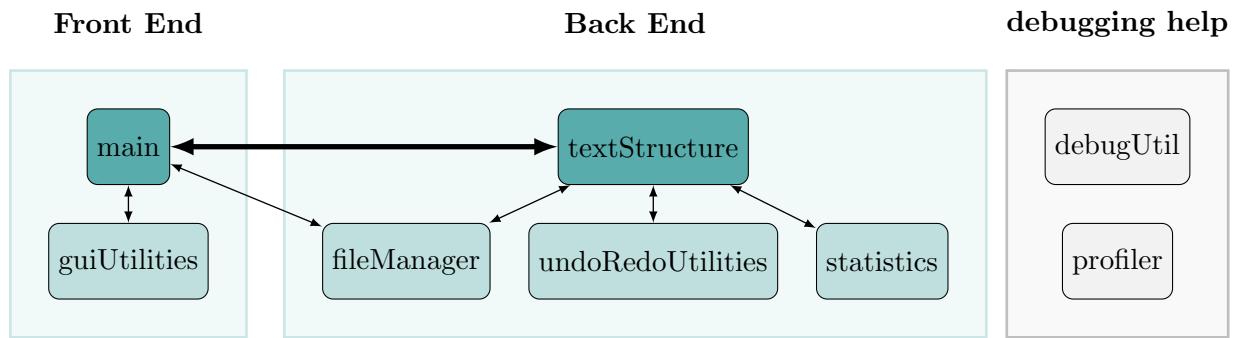
The `textStructure.h` header is the main interface connecting the GUI in `main.c` and the actual text data structure implementation in `textStructure.c`.

`print_items_after` is one of our most important parts and our method to that displays text in our terminal. Prints a certain number of lines starting from a chosen atomic position in our text sequence. Checks if sequence exists and line break standards. If good proceeds. It walks through blocks of text data and handles things like UTF-8 character boundaries, skips control characters and detects line breaks or end-of-block to finish a text line.

Changes current line segment to wide-character string for terminal compatibility. Output is the processed string that goes to the terminal at correct screen position.

For efficiency it only prints lines that are actually visible on the terminal, so out of view lines don't get printed. For that we have a variable that has the absolute position of the line at the

Figure 1: Project Code Structure



top of the screen (e.g. top line is actually the 5th line in the entire text). Also this is where the lineStats get update and if a line goes across multiple blocks the counters(`atomicsInLine`, `nbrOfUtf8CharsNoControlCharsInLine`) carry over to next block

In `guiUtilities` we have our line statistics like what is the current line number at the top of the screen or how many chars are in the specified line. The methods in here are used for things like scrolling, jumping to a specific line when using the search function or managing the line stats. Standouts are the converter from UTF-8 to wide characters and a way to translate cursor position to a position in our data structure.

Our Cursor refreshs independent of our text. That means when our cursor moves position it doesn't cause the text to also be refreshed, that would be a big performance hit.

4 Results

```
Hello
This
Is
a Test

Ln 5-5, Col 1-7 || Line breaks: LINUX || 5 words, 5 lines || Ctrl-l to quit
[Save] [Search] [S&R]
```

Figure 2: GUI of our text editor running on WSL

Above you can see our GUI. From the top left you can see 5 lines written, one of the blank. 'a Test' is marked and can be use to copy/paste or delete this section. Ln 5-5 means marked are rows from 5 to 5, same for Col but column 1 to 7. 'Line breaks: LINUX' means that the current line break style used is LINUX (`\n`). It also supports MSDOS (`\r\n`) and MAC (`\r`). Next to it you also have total line and total word count and to the right is the short-cut to exit the editor. At the very bottom are buttons that you can press to use them. S&R means search and replace.

All aspctes of our GUI will be shown in this video [3], as some of these things are impractical

to show on a picture.

5 Discussion

We have most normal features a text editor should have like a cursor, mouse integration, deleting, line breaks etc. We even have some more "esoteric" features like search/replace, copy/past and can even open multi GB files, so in terms of actual features we are pretty satisfied with the outcome.

But there are also some problems that could use some work, for example emojis are displayed as more than one character even though they are only one UTF-8 character, so they are cumbersome to use in the editor and can cause bugs.

Horizontal scrolling is also not working in the final build, it used to work at some point but stopped working somewhere down the line. Now when you reach the right end of the screen it just stops.

6 Conclusion

7 Lessons Learned

For the GUI aspect we learned that cursor and internal position can very easily desync, it happened way too often and not even because of wide character support.

The other big part for the GUI group was also time management, we were way too slow at the beginning and middle of the project and definitely caused the other group some headaches.

Next time we should definitely try to put a lot more work into the beginning.

References

- [1] Thomas E. Dickey, "ncurses." [Online]. Available: <https://invisible-island.net/ncurses/>. Accessed 10 Jun. 2025.
- [2] astrand & Contributors, "xclip: Command line interface to the X11 clipboard." [Online]. Available: <https://github.com/astrand/xclip>. Accessed 10 Jun. 2025.
- [3] Jorn Riedel, "Demo of Text Editor Project." [Online]. Available: https://drive.google.com/file/d/1obnfqQgP9Gp3SvdBcAY_Rjes19-MmlR8/view?usp=drive_link. Accessed 10 Jun. 2025.

Appendix A: Plots

Appendix B: Materials

B.1 Software Requirements

Requirements we set ourselves at the initial project submission:

- custom data structure
- possibility to open files
- utf8 support
- line break styles
- move cursor
- clickable buttons
- selectable text
- edit text
- special functions: find, replace
- copy ,paste
- handle different line break standards
- word count statistic
- line break type statistic

Additional requirements we managed to do:

- cursor position statistic
- line statistic
- replace all occurrences

Appendix C: Declaration of Independent Authorship

Copy text from https://dmi.unibas.ch/fileadmin/user_upload/dmi/Studium/Computer_Science/Diverses/Verwendung_AI/2025-02-17_Eigenstaendigkeitserklaerung_Declaration-of-Independent-Authorship_DE_EN_neu.pdf

And sign it (all authors!)

Signature of all authors as PDF

Sig Ath

Sig Ath

Sig Ath

Sig Ath