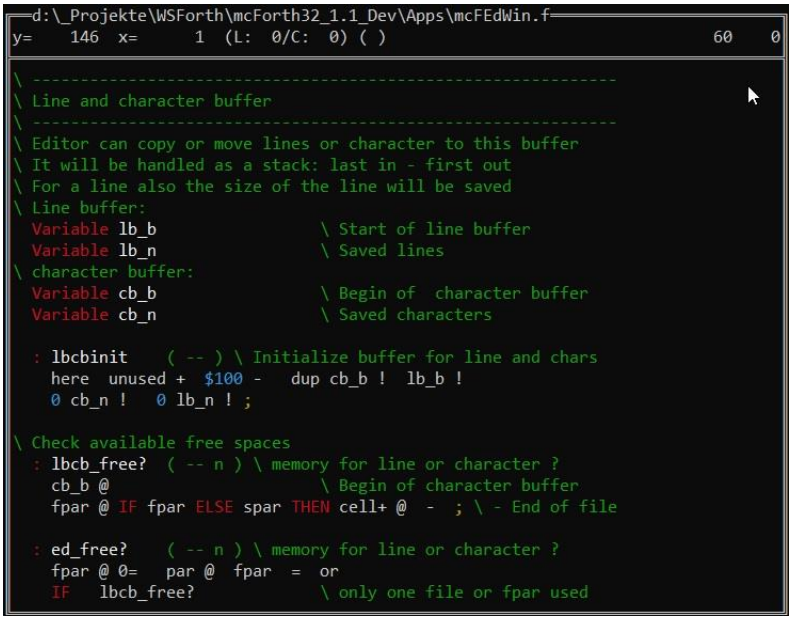
Klaus Kohl-Schöpe

Screen- und sequenzieller Editor in Forth

Schon vor über 30 Jahren hatte ich einen Screen-Editor für mein KKForth realisiert. Mein neueres mcForth kann aber auch mit sequenziellen Files umgehen und deshalb wollte ich ein Editor, der mir auch da bei Fehler die richtige Stelle zeigt. Als kleiner Bonus wurde dann auch noch das Markdown-Feature eingebaut, was mir die wichtigsten Forth-Worte farbig anzeigt.



```
d:\_Projekte\WSForth\mcForth32_1.1_Dev\Apps\mcFEdWin.f
y=    146  x=      1  (L:  0/C:  0) ( )                                  60    0

\ ------------------------------------------------------------
\ Line and character buffer
\ ------------------------------------------------------------
\ Editor can copy or move lines or character to this buffer
\ It will be handled as a stack: last in - first out
\ For a line also the size of the line will be saved
\ Line buffer:
  Variable lb_b                \ Start of line buffer
  Variable lb_n                \ Saved lines
\ character buffer:
  Variable cb_b                \ Begin of  character buffer
  Variable cb_n                \ Saved characters

  : lbcbinit    ( -- ) \ Initialize buffer for line and chars
    here  unused +  $100 -   dup cb_b !  lb_b !
    0 cb_n !   0 lb_n ! ;

\ Check available free spaces
  : lbcb_free?  ( -- n ) \ memory for line or character ?
    cb_b @                       \ Begin of character buffer
    fpar @ IF fpar ELSE spar THEN cell+ @  -  ; \ - End of file

  : ed_free?    ( -- n ) \ memory for line or character ?
    fpar @ 0=   par @  fpar  =  or
    IF   lbcb_free?              \ only one file or fpar used
```

Wünsche

Die Vorgabe war, dass alle Features des Screen-Editors vorhanden und auf sequenzielle Files ausgedehnt werden:

- Einstellbare Fenstergröße
- Einstellbare Screengröße
- Zeichen- und Zeilenpuffer
- Start bei bestimmter Zeile/Position
- Suchfunktion
- Markdown-Support für Forth

Da ich meinen ersten Editor auch bei Meßsysteme mit 8*40 Zeichen Display verwendet habe, war ein scrollen der Anzeige wichtig. Außerdem sollten nicht nur die üblichen 16*64 Zeichen Screens, sondern auch die C64-Version mit 25*40+24 Zeichen anzeigbar sein. Ein getrennter Puffer für Zeichen und Zeilen erlaubt das Kopieren bzw. Verschieben von Textteilen. Natürlich gehört zu einem Editor auch Suchen und Ersetzen von Texten.

Nachdem der Editor funktionierte, dachte ich mir, warum nicht gleich noch das Markdown einzubauen, um Zahlen und Strukturen besser zu sehen.

Noch nicht vollständig realisiert aber schon vorbereitet ist das Handling eines zweiten Files was bei Nutzung zu einer Aufspaltung des verfügbaren Speichers (freier Bereich über heap) führt.

Verwendung

Die Verwendung ist einfach, da der Editor in mcForth das aktuell offene File verwendet und mir bei **`v`** die aktuelle Fehlerposition und bei **`l`** (line --) die angegebene Zeile zeigt.

Gforth erwartet Zeile und Filename:

line s“ Filename“ **`edit`**

Für die Tastaturbelegung am besten das Sourcefile ansehen (oder evtl. ändern).

Anpassung an Gforth

Das mcFEdWin.f ist auf meinem mcForth für Windows entstanden. Deshalb hat es einige Besonderheiten, die ich für das ebenfalls verfügbare GforthEdWin.f (hier auch für Windows) „eliminieren“ musste:

- Nutzung von Create / End-Create
- Getrenntes Handling von Flash / RAM (**c@p** statt **c@** …)
- Andere Filebefehle und Tastencodes
- Farben über ESC-Sequenz

Gerade wegen der unterschiedlichen Tastencodes sind einige Funktionen nicht genutzt, können aber leicht wieder aktiviert werden. Leider sind auch die Farben und Zeichen für die Rahmen anders. Aber mein Ziel war hier nicht eine vollständige Anpassung sondern ein funktionierendes Grundgerüst auf ANS Forth Basis.

Implementierung

Ich hoffe, dass dieser Editor mit den vielen Kommentaren gut zu verstehen und zu ändern ist, da fast alle Parameter wie Bildschirm-/Screen-Größe, verwendete Rahmenzeichen und Farben als Konstanten bzw. als Value (wenn veränderbar) definiert wurden.

Da mein alter Screeneditor mit zwei Files parallel umgehen konnte, sind noch Reste davon in der Source vorhanden. Deshalb sind alle mit **sv_**... beginnenden Variablen für File-ID, Speicher-, Bildschirm- und Cursorposition in einem Datenfeld gespeichert, auf dass **par** zeigt und entweder bei **spar** oder **fpar** liegen kann. Da aktuell nur ein File verwendet wird (**sv_fid** in **fpar** ist 0), wird der gesamte freie Speicher für das File und den Zeichen-/Zeilenpuffer verwendet. Dies wird aber nur bei der Abfrage des freien Speichers für das File oder den Puffer gebraucht.

Das File wird mit **loadfile** komplett in das interne RAM oberhalb von **here** geladen, Tab durch Space ersetzt und danach geprüft, ob es ein Screenfile (n * **#c/s** Zeichen ohne CR bzw. CR/LF) oder ein sequenzielles File ist. Das Ergebnis wird dann in **sv_seq?** gespeichert, welches mit Wert 0, 1 oder 2 auch die Anzahl der Zeichen am Ende einer Zeile zeigt. Da eine Mischung von CR und CR/LF nicht erlaubt ist, wird eine Fehlermeldung ausgegeben. Am Ende eines Files wird bei Bedarf CR bzw. CR/LF ergänzt und damit schon das File geändert. Dies ist sichtbar als (*) in der 2.ten Zeile des Editors hinter Position und Anzahl Zeichen/Zeilen im Puffer. Die Größe des Fensters für den Editor wird hier auch mit 80*25 für sequenzielle Files oder dem Screenformat 16*64 (**#l/s** und **#c/l**) festgelegt.

Der Rahmen des Editors wird mit **frame** aufgebaut und mit **lcdrestore** gefüllt, wobei einzelne Bits in **disp?** entscheiden, ob das ganze Fenster, der Teil ab dem Cursor oder nur die aktuelle Zeile aktualisiert wird. Deshalb wird vorher noch geprüft, ob der Cursor überhaupt noch im Fensterbereich ist und evtl. der Ausschnitt korrigiert. Danach wird der Cursor gesetzt und dessen Position in der 2. Zeile aktualisiert.

Der eigentliche Start des Editors ist **edit** (mcForth) bzw. **(edit** (GForth), welches das File ladet, die gewünschte Stelle (Zeile oder Position im File) sucht, den Bildschirm aufbaut und dann entsprechend der gedrückten Tasten reagiert. Mit ESC wird der Editor verlassen, wobei bei Änderung gefragt wird, ob das File gespeichert werden soll. Die verwendeten Tastencodes sind als Konstanten und die Funktionen bei **eseq** als lange Tabelle abgelegt, welche durch **sequencer** gesucht und ausgeführt wird.

Die Funktionen selbst wie Cursorbewegung, Einfügen oder Löschen, Handling des Zeichen-/Zeilen-Puffers, Suchen und Ersetzen, Zusammenfügen oder Aufteilen von Zeilen oder Wechsel von Groß- zu Kleinschrift (und umgekehrt) dürfte in den Sourcen leicht zu verstehen sein. Ein Überbleibsel aus dem alten Editor ist noch

das Einfügen einer ID – also Datum mit bis zu 4 Zeichen Kennung. Mit F10 kann ich diese vorgeben und mit F8 wird dann in Klammern diese ID an der Cursorpostion eingesetzt.

An vielen Stellen wird man den Einfluss von **`sv_seq?`** feststellen, da damit nicht nur die nicht druckbaren Zeichen am Zeilenende festgelegt wird, sondern sorgt auch dafür, dass man beim Scrollen nicht aus dem aktuellen Screen herauskommt. Da der Editor immer im Insert-Mode ist, wird beim Einfügen in Screenfiles auch geprüft, ob noch genügend Zeichen am Zeilenende frei sind bzw. ob die letzte Zeile leer ist.

Markdown

Bei der Ausgabe einer Zeile kommt in **`.line`** auch das Markdown zum Einsatz, dass damit nur auf diese Zeile begrenzt ist. Selbst wenn der Anfang der Zeile nicht sichtbar ist, muss dieser beachtet werden um die aktuelle Farbe zu ermitteln. Dies geschieht mit einer Statemaschine die sich merkt mit welcher Farbe bis zu welcher Adresse ein Text ausgegeben wird und ob danach Leerzeichen übersprungen oder nach einem bestimmten Zeichen gesucht werden soll.

Deshalb enthält die Tabelle für das Marking neben dem Befehlsstring den nächsten State, die Farbe für den gefundenen String, die Farbe für das evtl. nachfolgende Wort oder String und das Zeichen, das diesen String beendet. Damit hat man die Möglichkeit, nicht nur das Wort selbst, sondern ganze Strings oder Kommentare wie **`\`** , **`\\`** oder **`(…)`** und nachfolgende Worte in unterschiedlichen Farben anzuzeigen.

Eine Besonderheit ist dann noch das Erkennen von Zahlen, die ebenfalls wie Strings in blauer Farbe angezeigt werden. Dabei ist mir aufgefallen, dass in Gforth das **`#`** auch schon als Zahl behandelt wird weil das verwendete **`s>number?`** eine 0 erkennt.

Wünschen

Wie immer, wenn man mit einem Projekt (fast) fertig ist, kommen weitere Wünsche auf, die man sich für das nächste Update (hoffentlich nicht wieder in 30 Jahren) merkt. Diese sind hier:

- Volle Implementierung des 2. Files
- Laden/ Speichern (andere Namen)
- Undo-Puffer
- Kopieren beliebiger Bereiche
- Kooperation mit Compiler
- Erweiterung zum Projektmanager
- Optimierung der Geschwindigkeit

Ideal ist es natürlich, wenn der Editor ähnlich wie Visual Studio Code gleich das ganze Projekt oder Verzeichnis verwaltet. Der Anfang wäre schon das Handling eines 2.ten Files und Laden/Speichern unter andere Namen. Ein kleiner Schritt in diese Richtung ist schon, wenn man Teile aus dem File direkt an den Compiler schicken könnte.

Man merkt deutlich den Aufbau des Bildschirms, was vermutlich größtenteils auf Markdown und der Suche nach der gewünschten Zeile geschuldet ist.

Copyright

Ich würde mich über häufige Verwendung und der Erwähnung der Quelle freuen. Die beiden Files und diese Anleitung in deutscher und bald auch englischer Version ist in meinem Github unter mcForth zu finden:

https://github.com/KlaKoSch/mcForth/tree/main/EdWin

Viel Spaß damit wünscht
Klaus Kohl-Schöpe