Programmieren in C

Prof. Dr. Peter Thiemann
Hannes Saffrich
Sommersemester 2021

Universität Freiburg
Institut für Informatik

Übungsblatt 6

Abgabe: Montag, 07.06.2021, 9:00 Uhr morgens

Hinweis: Vergessen Sie nicht ein Treffen mit Ihrem Tutor zu vereinbaren!

Ein Programm mit einem *Graphical User Interface (GUI)* stellt seinen Zustand graphisch in einem Fenster dar mit dem der Benutzer über Maus und Tastatur interagieren kann. Ein Beispiel hierfür ist der Texteditor `vscode` aus Abbildung 1.

Ein Programm mit einem *Terminal User Interface (TUI)* verhält sich wie mit einem Graphical User Interface, verwendet aber statt einem eigenen Fenster das Terminal und ist daher rein text-basiert. Ein Beispiel hierfür ist der Texteditor `vim` aus Abbildung 2.

Im Gegensatz zu den interaktiven Kommandozeilen-Programmen, die wir bisher geschrieben haben, erlaubt es ein TUI die Zeichen im Terminal an beliebigen Stellen zu verändern, anstatt nur am Ende neuen Text anzuhängen.

TUIs sind simpler zu implementieren wie GUIs und funktionieren auch wenn man sich mit `ssh` auf einem Server einloggt. Server haben meistens keine Graphische Benutzeroberfläche installiert (um Ressourcen zu sparen) und unterstützen daher auch keine GUI Programme. Es kann also nützlich sein mindestens einen Texteditor mit einem TUI benutzen zu können.

In diesem Übungsblatt sollen Sie eine Library implementieren, die es erlaubt Programme mit einem Terminal User Interface zu schreiben.

Im nächsten Übungsblatt kommt dann eine Anwendung von dieser Library:

https://www.youtube.com/watch?v=H4KgUTdogHQ

(⌐■_■)

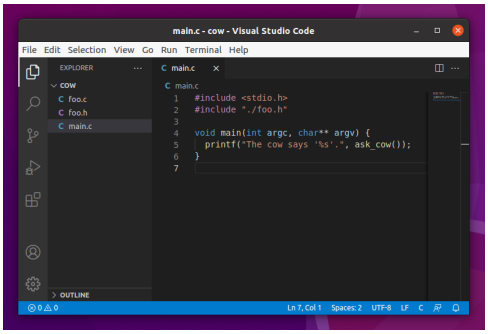

Abbildung 1: `vscode` (GUI)

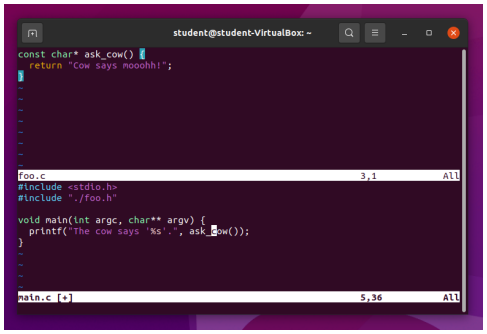

Abbildung 2: `vim` (TUI)

ANSI Escape Codes

Terminals unterstützen sogenannte *ANSI escape codes*. Diese sind Strings, die mit dem Zeichen `'\e'` beginnen. Wird ein solcher String mit `printf` ausgegeben, so zeigt das Terminal diesen nicht als Text an, sondern führt bestimmte Befehle aus.

Für unsere TUI-Library sind vier Arten von ANSI escape codes interessant:

- Codes zum Ändern der Text- oder Hintergrund-Farbe die für zukünftige Ausgaben verwendet wird;
- Codes die die Cursor-Position verändern, sodass zukünftige Ausgaben an einer anderen Koordinate im Terminal angezeigt werden;
- Codes die das Anzeigen der aktuellen Cursor-Position ein- oder ausschalten;
- Codes die den gesamten Inhalt des Terminals löschen.

Der Code `"\e[0;31m"` setzt zum Beispiel die Text-Farbe auf rot und der Code `"\e[0m"` setzt Text- und Hintergrund-Farbe wieder auf die Standard-Einstellungen zurück. Die Anweisung

```
printf("I'm default text\e[0;31mI'm red text\e[0mBack to default text.\n");
```

erzeugt also die Ausgabe

```
I'm default textI'm red textBack to default text.
```

Da es verwirrend ist die Codes direkt innerhalb von Strings anzugeben, versteckt man die Codes im Regelfall hinter Macros:

```
#define FG_RED      "\e[0;31m"
#define COLOR_RESET "\e[0m"
```

Statt der vorherigen Anweisung kann man dann

```
printf("I'm default text" FG_RED "I'm red text" COLOR_RESET
       "Back to default text.\n");
```

oder

```
printf("I'm default text%sI'm red text%sBack to default text.\n",
       FG_RED, COLOR_RESET);
```

schreiben.

Die Effekte der Codes bleiben auch über mehrere `printf`-Anweisungen erhalten. Die Anweisungen

```
printf("I'm default text");
printf(FG_RED);
printf("I'm red text");
printf(COLOR_RESET);
printf("Back to default text.\n");
```

haben also den gleichen Effekt wie die vorherigen.

Aufgabe 6.1 (Make friends with ANSI; 2 Punkte)

In Ihrem Repository befinden sich in `blatt06` bereits einige `.c`- und `.h`-Dateien. Wie im vorherigen Übungsblatt sollen Sie diese vervollständigen. Eine vollständige `Makefile` ist dieses mal auch schon mit dabei.

In der Datei `ansi_codes.h` finden Sie alle für uns relevanten ANSI escape codes. Diese sind als Macros definiert mit Ausnahme der Codes zum Bewegen des Cursors: für diese gibt es in der `ansi_codes.h` eine Funktion

```
move_cursor_to(size_t x, size_t y)
```

die mit `print` den Code ausgibt um den Cursor in Spalte `x` und Zeile `y` zu bewegen, sodass nachfolgende Ausgaben an dieser Stelle angezeigt werden.

Standardmäßig verwenden Terminals *Line Buffering*, d.h. Ausgaben werden erst dann im Terminal angezeigt, wenn ein Zeilenumbruch `'\n'` ausgegeben wird. Wenn wir aber mit ANSI escape codes den Cursor in vorherige Zeilen bewegen, dann wollen wir meistens *keinen* Zeilenumbruch ausgeben, da dieser die nachfolgenden Zeilen zerstören würde. Stattdessen kann man `fflush(stdout);` verwenden, welches die bisherigen Ausgaben auch ohne ein `'\n'` auf dem Terminal ausgibt. Die Funktion `fflush` und die globale Variable `stdout` sind in `stdio.h` deklariert.[1]

Schreiben Sie in der `ansi_example.c` ein Programm, welches **ohne ein einziges `'\n'` zu verwenden** folgendes Verhalten zeigt:

- als erstes wird ein ANSI code ausgegeben, der den bisherige Inhalt des Terminals löscht (siehe `ansi_codes.h`);
- dann werden in den ersten 3 Zeilen jeweils 20 Hash-Zeichen `'#'` ausgegeben;
- dann werden die Hashs ab Zeile 2 Spalte 6 mit einem roten `hello` überschrieben;
- dann wird der Cursor zum Ende der Ausgabe in Zeile 4 Spalte 1 bewegt.

Nach dem Ausführen Ihres Programms, soll das Terminal wie folgt aussehen:

Der letzte Schritt ist notwendig, da nach dem Programmende der ausgegebene Text, der nach der aktuellen Cursorposition kommt, nicht mehr zu sehen ist.

Da die anderen Programme zu diesem Zeitpunkt noch unvollständig sind, bietet es sich an `make ansi_example` statt `make compile` zu verwenden.

Hinweis: Es bietet sich an dieser Stelle an etwas mit den ANSI escape codes herumzuspielen um ein Gefühl für deren genaue Verhaltensweise zu bekommen.

[1] `fflush` steht für *file flush* und schreibt den Cache einer Datei auf die Festplatte; `stdout` ist eine Datei, die mit den Terminal verbunden ist und wird auch von `printf` intern beschrieben.

Performance und Matrizen

Bei dem Spiel, dass wir uns im nächsten Übungsblatt vornehmen werden, müssen bei jedem Zeitschritt die Asteroiden und die vom Raumschiff abgefeuerten Projektile bewegt werden (siehe Video).

Eine einfache Möglichkeit zum Bewegen eines Projektils wäre z.B. das Zeichen an der vorherigen Position des Projektils auf `' '` zu setzen und das Zeichen an der aktuellen Position des Projektils auf `'>'` zu setzen:

```
move_cursor_position(old_x, old_y);
printf(" ");
move_cursor_position(new_x, new_y);
printf(">");
```

Es wäre aber deutlich angenehmer, wenn man bei jedem Zeitschritt einfach das gesamte Spielfeld erneut ausgeben könnte und sich keine Gedanken machen müsste welche Zeichen vorher im Terminal angezeigt wurden.

Leider sind die meisten Terminals aber ziemlich langsam wenn es darum geht in kurzer Zeit große Mengen an Ausgaben zu verarbeiten, was dann zu einem ruckelnden Bild und anderen Anzeigefehlern führt.

Wir wollen unsere TUI Library daher so programmieren, dass man beim Benutzen der Library so tun kann als würde man jedes mal das gesamte Spielfeld neu zeichnen, aber die Library sich im Hintergrund merkt, welche Zeichen vorher im Terminal zu sehen waren und nur die Zeichen erneut ausgibt, die sich verändert haben.

Hierzu müssen wir uns zunächst überlegen, wie wir den Inhalt des Terminals mit einer Datenstruktur in C repräsentieren können:

- An jeder Stelle im Terminal steht ein Zeichen mit einer bestimmten Text- und Hintergrund-Farbe:

  ```
  typedef struct Cell {         /* from tui_matrix.h */
    char content;                 /* The character at this position */
    const char* text_color;       /* ANSI Code */
    const char* background_color; /* ANSI Code */
  } Cell;
  ```

- Die Zeichen im Terminal sind in einer Matrix angeordnet. Da sich die Größe des Terminalfensters verändern kann, reicht hierfür kein 2D-Array von fester Größe, sondern wir müssen eine Datenstruktur analog zu `Vec` definieren:

  ```
  struct Matrix {               /* from tui_matrix.c */
    Cell* cells;
    size_t width;
    size_t height;
  };
  ```

Unsere TUI Library soll zwei dieser Matrizen verwalten: eine Matrix `old` für den aktuellen Inhalt des Terminals und eine Matrix `new` für den zukünftigen Inhalt des Terminals.

Wenn man die Library benutzt, z.B. um das Spielfeld zu zeichnen, dann sollen die Zeichen nicht direkt im Terminal ausgegeben werden, sondern an der richtigen Stelle in `new` eingetragen werden. Nach dem Zeichnen des Spielfelds ruft man dann eine Funktion auf, die `old` und `new` vergleicht, die Zeichen im Terminal ausgibt die sich verändert haben und `old` so anpasst, dass es danach die gleichen Zeichen enthält wie `new`.

Aufgabe 6.2 (Terminal-Matrix; 8 Punkte)

Die Funktionen für die oben beschrieben Matrizen sind in der `tui_matrix.h` deklariert. Definieren Sie diese Funktionen in der `tui_matrix.c` und schreiben Sie sinnvolle Unittests in der `tui_matrix_test.c`.

Für `matrix_print_update` können keine Unittests geschrieben werden. Um zu überprüfen ob Sie diese zumindest teilweise korrekt implementiert haben, wird eine `tui_matrix_example.c` bereitgestellt.

Punkteverteilung:

- 1 Punkt für `matrix_new, matrix_free`;
- 1 Punkt für `matrix_width, matrix_height, matrix_cell_at`;
- 1 Punkt für `matrix_clear_with` und `matrix_clear`;
- 1 Punkt für `matrix_set_str_at`;
- 2 Punkte für `matrix_resize`;
- 2 Punkte für `matrix_print_update`.

Die eigentliche Library

Eine Library für TUIs muss Funktionen für drei verschiedene Aufgabenbereiche zu Verfügung stellen:

1. Verändern der Zeichen im Terminal an beliebigen Stellen;
2. Abfragen ob eine Taste gedrückt wurde und wenn ja welche Taste;
3. Abfragen welche Größe das Terminals aktuell hat, sodass man mitbekommt wenn das Terminalfenster vergrößert oder verkleinert wurde.

In den Dateien `tui_io.h`, `tui_io.c`, `tui.h`, `tui.c` ist diese Funktionalität bereits implementiert. Für den 1. Bereich werden hierzu die Funktionen aus `tui_matrix.h` verwendet. Für den 2. und 3. Bereich stellen wir die zugehörigen Funktionen in der `tui_io.h` und `tui_io.c` bereit.

Da ein Programm immer mit genau einem Terminal verbunden ist, werden die Matrizen `new` und `old` in der `tui.c` als globale Variablen verwaltet. Dies erlaubt es beim Benutzen der Library, z.B. statt

```
Cell* matrix_cell_at(Matrix* m, size_t x, size_t y);
```

die Funktion

```
Cell* tui_cell_at(size_t x, size_t y) {
  return matrix_cell_at(new, x, y);
}
```

zu verwenden, die die Existenz der Matrizen vor dem Benutzer der Library versteckt.

Machen Sie sich mit den Funktionen in `tui.h` und `tui_io.h` vertraut und versuchen Sie zu verstehen wie die `tui.c` implementiert ist. Die `tui_io.c` enthält Schwarze Magie, die Sie zu diesem Zeitpunkt noch nicht verstehen müssen.

In der `tui_example.c` finden Sie folgendes Beispiel:

```
#include <unistd.h> /* Provides the `usleep` function used below. */
#include "./tui.h"  /* Transitively includes tui_io.h and ansi_codes.h */

int main(void) {
  tui_init();    /* Initialize the matrices and set up the terminal */
  char c = '!';  /* The char to display in the middle of the screen */
  size_t t = 0;  /* The current loop iteration (t for time) */

  while (1) {
    if (stdin_has_changed()) {  /* Check if the user has pressed a key. */
      c = read_from_stdin();    /* Retrieve the key pressed by the user. */
      if (c == 'q')             /* Quit the program if 'q' has been pressed. */
        break;
    }

    Size2 size = tui_size(); /* Get the current terminal size and resize
                                the `new` and `old` matrices if necessary. */
    tui_clear();             /* Clear the `new` matrix. */

    /* Change the `new` matrix, such that the cell in the middle of the screen
       contains the char c. Adding (t % 3) makes the character move a bit around
       each iteration, because movement is exciting :) */
    Cell* cell = tui_cell_at(size.x / 2 + (t % 3), size.y / 2);
    cell->content = c;
    cell->text_color = FG_RED;
    cell->background_color = BG_BLACK;

    tui_update();            /* Print the changes we did to the terminal */
    t += 1;
    usleep(100000);          /* Wait for 0.1 seconds before continuing
  }                             with the next iteration. */

  tui_shutdown();            /* Free memory and bring the terminal back into
  return 0;                     normal mode. */
}
```

Aufgabe 6.3 (Rainbows!; 4 Punkte)

Schreiben Sie in der `rainbows.c` ein Programm, welches ein Kommandozeilenargument entgegen nimmt und die TUI Library verwendet um den Text in Regenbogenfarben über den Bildschirm laufen zu lassen: https://youtu.be/eDnGVXZfwnY

Was die genauen Farben und Darstellung angeht, können Sie auch gerne vom Demo-Video abweichen, solange die folgenden Kriterien erfüllt sind:

- Die Darstellung muss damit zurecht kommen, wenn die Größe des Terminalfensters verändert wird.
- Der Text muss sich bewegen.
- Wenn ein Teil des Texts über den Bildschirmrand hinausragt, so muss dieser korrekt wieder am Anfang der Zeile angezeigt werden (wie im Video).

Aufgabe 6.4 (Erfahrungen; 2 Punkte)

Notieren Sie Ihre Erfahrungen mit diesem Übungsblatt in der Datei `erfahrungen.txt` (benötigter Zeitaufwand, Probleme, Bezug zur Vorlesung, Interessantes, etc.).

Der Zeitaufwand *muss* dabei in der ersten Zeile und in exakt dem folgenden Format notiert werden, da wir sonst nicht automatisiert eine Statistik erheben können:

```
Zeitaufwand: 3:30

<...Andere Erfahrungen...>
```

Die Angabe `3:30` steht hier für 3 Stunden und 30 Minuten.