6. ANSI-Steuersequenzen

6.1. Allgemeines

Dieses Kapitel greift auf Elemente zurück, die erst später im Skript erläutert werden. Für den Einstieg sind nur die ANSI-Control-Sequenzen zur Steuerung der Cursorposition und zum Löschen des Bildschirms interessant. Der Rest dieses Kapitels kann beim ersten Lesen übersprungen werden.

Ein Computer bildet die uns vertrauten Zeichen (Ziffern, Buchstaben, Sonderzeichen) auf ein eindeutiges Bitmuster ab, mit dem er intern arbeitet. Es muss folglich Vorschriften geben, die besagen, durch welchen Wert ein bestimmtes Zeichen im Speicher repräsentiert wird. Eine solche Vorschrift - Kode genannt - bildet die Zeichen eines Zeichenvorrats (z.B. den der Menschen) auf die Zeichen eines zweiten Zeichenvorrats (z.B. den des Rechners) ab.

Einer der gebräuchlichsten Kodes ist der vom American National Standards Institut (ANSI) definierte American Standard Code for International Interchange (ASCII). In der Tabelle im Anhang sind alle ASCII-Zeichen mit den zugehörigen Werten ersichtlich. Dieser Kode ist ein 7-Bit-Kode, d.h. ASCII nutzt von den insgesamt 8 Bits eines Bytes nur 7 Bits für die interne Darstellung eines Zeichens aus. Mit 7 Bits lassen sich $2^7 = 128$ Kombinationen darstellen, wobei jeder einzelnen Kombination genau ein Zeichen zugeordnet ist. Später wurde der erweiterte ASCII-Code, bei dem alle 8 Bits eines Bytes ($2^8 = 256$ verschiedene Kombinationen) verwendet werden, eingeführt. Der Zeichenvorrat setzt sich aus darstellbaren Zeichen (dezimale ASCII-Werte von 32 bis 126 und von 128 bis 254) und Steuerzeichen (dezimale ASCII-Werte von 0 bis 31 und 127) zusammen. Darstellbare Zeichen erscheinen auf dem Bildschirm in Form eines Pixelmusters. Im Gegensatz dazu sind den Steuerzeichen kontrollierende und steuernde Aufgaben zugeordnet. Die meisten dieser Steuerzeichen spielen zwar in der Datenübertragung eine Rolle, haben allerdings auf die Steuerung des Bildschirms keinen Einfluss. Werden diese an den Bildschirm geschickt, so wird das zugeordnete Zeichen ausgegeben, z.B. wird der ASCII-Wert 1 an den Bildschirm geschickt, erscheint ein Smiley-Gesicht. Die folgenden Steuerzeichen führen dagegen konkrete Aktionen aus, wenn sie an den Bildschirm geschickt werden.

ASCII-Wert		**Bezeichnung**	**Aktion**
dez.	**hex.**		
7	0x07	BEL (Bell)	erzeugt ein akustisches Signal

8	0x08	BS (Backspace)	Bewegt den Cursor um eine Position nach links. Falls sich der Cursor bereits am Zeilenanfang befindet, bleibt dieses Steuerzeichen wirkungslos.
9	0x09	HT (Horizontal Tab)	Positioniert den Cursor auf die nächste Tabulatormarke oder an den rechten Bildschirmrand, falls keine weiteren Tabulatormarken vorhanden sind.
10	0x0A	LF (Linefeed)	Setzt den Cursor an den Anfang der nächsten Zeile.
13	0x0D	CR (Carriage Return)	Verschiebt den Cursor an den Anfang der aktuellen Zeile.
26	0x1A	SUB (Substitute)	Kennzeichnet unter MS-DOS das Ende einer Datei.

Für die dezimalen ASCII-Werte 0 und 255 wird jeweils ein Leerzeichen ausgegeben. Für das Escape-Zeichen ESC (dezimaler ASCII-Wert 27) erfolgt keinerlei Ausgabe. Es spielt allerdings eine wichtige Rolle bei den Escape- und Control-Sequenzen. Mit ihnen können erst beispielsweise die Cursorposition gesteuert oder Bildschirmattribute gesetzt werden.

6.2. Escape- und Control-Sequenzen

Bei den **Escape- und Control-Sequenzen** handelt es sich um eine Folge von Zeichen, die alle mit dem **Escape-Zeichen ESC** beginnen. Mit diesen Zeichenfolgen ist es möglich, den Cursor gezielt zu positionieren, Bildschirmattribute zu setzen oder die Tastaturbelegung zu manipulieren usw. Diese Zeichenfolgen erscheinen nicht auf dem Bildschirm, sondern üben steuernde und kontrollierende Aufgaben aus. ANSI hat die Escape- und Control-Sequenzen erstmals in den Standards X3.41-1974 und X3.64-1979 definiert. In der Praxis werden solche Zeichenfolgen meistens (fälschlicherweise) nur als Escape-Sequenzen bezeichnet. ANSI unterscheidet jedoch zwischen Escape- und Control-Sequenzen.

In diesem Abschnitt werden die Escape- und Control-Sequenzen nur allgemein vorgestellt und im nächsten Abschnitt werden einige ANSI-Control-Sequenzen und deren Umsetzung in C vorgestellt.

Escape-Sequenzen

Das Format einer Escape-Sequenz hat den folgenden allgemeinen Aufbau:

ESC	**I...I**	**F**
27	32-47	48-126
escape sequence introducer (1 Zeichen)	intermediate characters (0 oder mehrere Zeichen)	final character (1 Zeichen)

Es wurden jeweils die englischen Bezeichnungen beibehalten. Die Zahlen in der zweiten Zeile stellen jeweils die dezimalen ASCII-Werte der möglichen Zeichen dar. Im folgenden werden die drei Bereiche genauer beschrieben.

escape sequenz introducer: Hierbei handelt es sich um das ESC-Zeichen mit dem dezimalen ASCII-Wert 27, mit dem die Escape-Sequenz eingeleitet wird. Zeichen, die danach folgen, werden als Teil der Escape-Sequenz betrachtet und nicht auf dem Bildschirm ausgegeben.

intermediate characters: Dies sind Zeichen aus dem dezimalen ASCII-Bereich von 32 bis 47, die als Teil der Escape-Sequenz automatisch gespeichert werden.

final character: Dieses Zeichen aus dem dezimalen ASCII-Bereich von 48 bis 126 schließt eine Escape-Sequenz ab. Die Kombination aus intermediate characters und final character spezifiziert die Aufgabe der Sequenz. Die durch die Escape-Sequenz festgelegte Operation wird ausgeführt und die nachfolgenden Zeichen erscheinen wieder auf dem Bildschirm, soweit es sich um darstellbare Zeichen handelt.

Beispiel:

Das folgende Beispiel zeigt die Escape-Sequenz für den Befehl Select Character Set (SCS):

```
ESC(A
```

Dabei ist `ESC` der escape sequence introducer, '`(`' das intermediate character und '`A`' das final character.
Wichtiger Hinweis: Zwischen den drei Bereichen dürfen keine Leerzeichen stehen!

Control-Sequenzen

Der Aufbau von Control-Sequenzen ist mit dem von Escape-Sequenzen vergleichbar.

CSI	**P...P**	**I...I**	**F**
27 91	48-63	32-47	64-126
control sequence introducer (2 Zeichen)	parameter characters (0 oder mehrere Zeichen)	intermediate characters (0 oder mehrere Zeichen)	final character (1 Zeichen)

Im folgenden werden die vier Bereiche genauer beschrieben.

control sequenz introducer: Hierbei handelt es sich um das Escape-Zeichen ESC (dezimaler ASCII-

Wert: 27) gefolgt von der eckigen Klammer auf ('['; dezimaler ASCII-Wert: 91). Diese beiden Zeichen zusammen leiten die Control-Sequenz ein. Zeichen, die danach folgen, werden als Teil der Control-Sequenz betrachtet und nicht auf dem Bildschirm ausgegeben.

parameter characters: Es wird zwischen zwei Arten von parameter characters unterschieden: numerische und selektive Parameter. Ein numerischer Parameter ist eine Dezimalzahl und besteht aus Ziffern aus dem dezimalen ASCII-Bereich von 48 bis 57. Für den selektiven Parameter kann nur ein Zeichen aus dem dezimalen ASCII-Bereich von 58 bis 63 verwendet werden. Treten in einer Control-Sequenz mehrere Parameter auf, müssen diese mit einem Semikolon (';') voneinander getrennt werden.

intermediate characters: Dies sind Zeichen aus dem dezimalen ASCII-Bereich von 32 bis 47, die als Teil der Control-Sequenz automatisch gespeichert werden.

final character: Dieses Zeichen aus dem dezimalen ASCII-Bereich von 64 bis 126 schließt eine Control-Sequenz ab. Zusätzlich macht dieses Zeichen eine Aussage darüber, ob es sich um eine ANSI-Standard-Control-Sequenz oder um eine sogenannte private Control-Sequenz - auf die nicht weiter eingegangen werden - handelt.

Beispiel:

Das folgende Beispiel zeigt die Control-Sequenz für den Befehl Cursor Position (CUP; Näheres dazu im nächsten Abschnitt!):

```
ESC[4;10H
```

Dabei ist `ESC[` der control sequence introducer, '`4;10`' die parameter characters (2 Parameter mit Semikolon getrennt) und '`H`' das final character. In dieser Control-Sequenz sind keine intermediate characters enthalten. **Wichtiger Hinweis**: Zwischen den Bereichen dürfen auch hier keine Leerzeichen stehen!

6.3. ANSI-Control-Sequenzen

Für die nachfolgend beschriebenen **ANSI-Control-Sequenzen** findet man häufig die Bezeichnung ANSI-Escape-Sequenzen, obwohl dies nicht korrekt ist. Ferner werden in diesem Abschnitt nur eine Teilmenge der von ANSI definierten Control-Sequenzen vorgestellt. Damit die ANSI-Control-Sequenzen funktionieren, muss unter DOS der Treiber `ANSI.SYS` installiert sein. In dem DOS-Fenster unter Windows sowie in der Konsole unter Linux sollte dies ohne weiteres funktionieren.

Zu jeder Control-Sequenz existiert eine von ANSI festgelegte mnemotechnische Abkürzung, die hier auch jeweils angegeben wird. Für die Realisierung unter C sind diese aber nicht relevant. Zu beachten ist, dass die Zeichen einer Sequenz in genau der gleichen Art (Groß- und Kleinschreibung) angegeben werden müssen. Ferner dürfen keine Leerzeichen innerhalb einer Sequenz stehen.

Die Sequenzen in diesem Abschnitt werden unterteilt in die Bereiche Cursor-Steuerfunktionen, Löschfunktionen, Grafikmodusfunktionen und Funktionen zur Änderung der Tastaturbelegung.

Cursor-Steuerfunktionen

Mit Hilfe der folgenden ANSI-Control-Sequenzen kann der Cursor auf dem Bildschirm positioniert werden. In DOS-Fenstern und Konsolen, in denen gescrollt werden kann, beziehen sie sich immer auf den aktuell sichtbaren Bereich.

1. Cursor Position

Sequenz: `ESC[P;PH` Mnemonik: `CUP` (Cursor Position)
oder
Sequenz: `ESC[P;Pf` Mnemonik: `HVP` (Horizintal and Vertical Position)

Die Control-Sequenzen `CUP` und `HVP` positionieren den Cursor auf eine beliebige Stelle auf dem Bildschirm. Der erste Parameter `P` legt die Zeile und der zweite die Spalte fest. Der Standardwert für fehlende Parameter ist 1, d.h. `ESC[H` ist identisch mit `ESC[1;1H`.

Die folgende Funktion zeigt, wie die beschriebene Control-Sequenz in C realisiert werden kann. Hierzu muss die Control-Sequenz an den Bildschirm geschickt werden. Dies erfolgt mit Hilfe der Funktion `printf`. Das Escape-Zeichen `ESC` wird dabei in oktaler Schreibweise ('`\033`') angegeben.

```
void position(int Zeile, int Spalte)
{  printf("\033[%d;%dH",Zeile,Spalte);
}
```

Der Funktionsaufruf `position(4,50);` bewegt den Cursor auf die fünfzigste Spalte in der vierten Zeile. Funktionsaufrufe erfolgen allerdings erst zur Laufzeit des Programms und verzögern somit seine Ausführung. Makros hingegen werden schon während der Übersetzung vom Präprozessor (siehe Kapitel *Präprozessorbefehle*) aufgelöst. Das heißt, dass Makros für gewöhnlich schneller abgearbeitet werden als Funktionen. Die Ausgabe der oben erläuterte Control-Sequenz als Makro lautet wie folgt.

```
#define POSITION(Ze, Sp)   printf("\033[%d;%dH",Ze,Sp)
```

Zusätzlich kann noch folgendes Makro definiert werden.

```
#define HOME               printf("\033[H")
```

Innerhalb des Programms könnten die Makroaufrufe wie folgt aussehen.

```
POSITION(23, 12);
HOME;
```

Es sei hier darauf hingewiesen, dass zwischen dem Makronamen und der linken öffnenden runden Klammer **kein** Leerzeichen stehen darf.

2. Cursor Up

Sequenz: `ESC[PA` Mnemonik: `CUU` (Cursor Up)

Diese Control-Sequenz bewegt den Cursor innerhalb der aktuellen Spalte um `P` Zeilen nach oben. Falls sich der Cursor bereits in der ersten Zeile des Bildschirms befindet, bleibt die Sequenz wirkungslos. Fehlt der Parameter `P`, so wird als Standardwert 1 angenommen.

```
#define UP(Anz)             printf("\033[%dA",Anz)
#define UP_LINE             printf("\033[A")
```

3. Cursor Down

Sequenz: `ESC[PB` Mnemonik: `CUD` (Cursor Down)

Diese Control-Sequenz bewegt den Cursor innerhalb der aktuellen Spalte um `P` Zeilen nach unten. Falls sich der Cursor bereits in der letzten Zeile des Bildschirms befindet, bleibt die Sequenz wirkungslos. Fehlt der Parameter `P`, so wird als Standardwert 1 angenommen.

```
#define DOWN(Anz)           printf("\033[%dB",Anz)
#define DOWN_LINE           printf("\033[B")
```

4. Cursor Forward

Sequenz: `ESC[PC` Mnemonik: `CUF` (Cursor Forward)

Diese Control-Sequenz bewegt den Cursor innerhalb der aktuellen Zeile um `P` Spalten nach rechts. Falls sich der Cursor bereits in der letzten Spalte der Zeile befindet, bleibt die Sequenz wirkungslos. Fehlt der Parameter `P`, so wird als Standardwert 1 angenommen.

```
#define RIGHT(Anz)          printf("\033[%dC",Anz)
#define ONE_POS_RIGHT       printf("\033[C")
```

5. Cursor Backward

Sequenz: `ESC[PD` Mnemonik: `CUB` (Cursor Backward)

Diese Control-Sequenz bewegt den Cursor innerhalb der aktuellen Zeile um `P` Spalten nach links. Falls sich der Cursor bereits in der ersten Spalte der Zeile befindet, bleibt die Sequenz wirkungslos. Fehlt der Parameter `P`, so wird als Standardwert 1 angenommen.

```
#define LEFT(Anz)           printf("\033[%dD",Anz)
#define ONE_POS_LEFT        printf("\033[D")
```

6. Save Cursor Position

Sequenz: ESC[s Mnemonik: SCP (Save Cursor Position)

Die Position des Cursors (Zeile und Spalte) zum Zeitpunkt der Ausführung der Control-Sequenz wird intern gespeichert. Mit Hilfe der RCP-Sequenz kann der Cursor auf die gespeicherte (alte) Position zurückgebracht werden.

```
#define STORE_POS          printf("\033[s")
```

7. Restore Cursor Position

Sequenz: ESC[u Mnemonik: RCP (Restore Cursor Position)

Der Cursor wird auf die zuvor mittels der SCP-Sequenz gespeicherten Position (Zeile und Spalte) gesetzt.

```
#define RESTORE_POS        printf("\033[u")
```

8. Device Status Report

Sequenz: ESC[6n Mnemonik: DSR (Device Status Report)

Die Control-Sequenz DSR fordert die aktuelle Position des Cursors an. Diese wird mit Hilfe der Control-Sequenz CPR in der Form ESC[P;PR in den Tastaturpuffer geschrieben. Von dort kann die Position des Cursors ausgelesen werden. Die erste Parameter P gibt die Zeile und der zweite die Spalte an. In dem anschließenden Beispiel wird gezeigt, wie die Control-Sequenzen DSR und CPR angewendet werden.

```
#define ACT_POS            printf("\033[6n")
```

9. Cursor Position Report

Sequenz: ESC[P;PR Mnemonik: CPR (Cursor Position Report)

Die Control-Sequenz CPR ist die Antwort auf die Control-Sequenz DSR und wird in der Standardeingabe (normalerweise im Tastaturpuffer) abgelegt. Dabei gibt der erste Parameter P die Zeile und der zweite die Spalte des Cursors an. Im folgenden Beispiel wird gezeigt, wie die Control-Sequenzen DSR und CPR angewendet werden.

In der folgenden Beispiel-Funktion Cursor_Position wird die aktuelle Cursorposition mit Hilfe des Makros ACT_POS (siehe oben) angefordert. Die Antwort wird aus dem Tastaturpuffer gelesen und an die aufrufende Funktion zurückgegeben. Die Funktion gibt zusätzlich einen Statuswert zurück, der angibt, ob ein Fehler aufgetreten ist oder nicht.

kap06_01.c

```
#include <stdio.h>
#include <stdlib.h>

#define FEHLER          0
#define OK              1

#define ESC             '\033'
#define LEFT_BRACKET    '['
#define NUL             '\0'
#define SEMIKOLON       ';'
#define FINAL_CHAR      'R'

#define ACT_POS         printf("\033[6n")

int Cursor_Position(int *Zeile, int *Spalte)
{
   int Zeichen, i;
   char Parameter1[5], Parameter2[5];

   // Control-Sequenz DSR: akt. Cursor-Position anfordern
   ACT_POS;

   // erstes Zeichen der CPR-Sequenz lesen (= ESC)
   if ((Zeichen = getch()) != ESC)
      return FEHLER;

   // zweites Zeichen der CPR-Sequenz lesen ('[')
   if ((Zeichen = getch()) != LEFT_BRACKET)
      return FEHLER;

   // ersten Parameter holen
   for (i = 0; isdigit(Zeichen = getch()); i++)
      Parameter1[i] = (char) Zeichen;
   Parameter1[++i] = NUL;

   // prüfen, ob Semikolon zwischen den Parametern steht
   if (Zeichen != SEMIKOLON)
      return FEHLER;

   // zweiten Parameter holen
   for (i = 0; isdigit(Zeichen = getch()); i++)
      Parameter2[i] = (char) Zeichen;
   Parameter2[++i] = NUL;

   // prüfen, ob letztes Zeichen gleich 'R'
   if (Zeichen != FINAL_CHAR)
```

```
      return FEHLER;

   // CPR-Sequenz ist ok -> Parameter übergeben
   *Zeile = atoi(Parameter1);
   *Spalte = atoi(Parameter2);
   return OK;
}

int main(void)
{
   int Zeile, Spalte, Sequenz_OK;

   // akt. Cursorposition holen
   Sequenz_OK = Cursor_Position(&Zeile, &Spalte);

   // Cursorposition ausgeben
   if (Sequenz_OK)
   {  printf("Akt. Cursorposition:\n");
      printf("Zeile:  %d\n", Zeile);
      printf("Spalte: %d\n", Spalte);
   }
   else
      printf("Akt. Position konnte nicht ermittelt
werden!");
   return 0;
}
```

Löschfunktionen

Die nachfolgend beschriebenen ANSI-Control-Sequenzen löschen bestimmte Bereiche des Bildschirms.

1. Erase Display

Sequenz: `ESC[2J` Mnemonik: `ED` (Erase Display)

Diese Sequenz hat das Löschen des kompletten (sichtbaren) Bildschirms zur Folge und setzt anschließend den Cursor in die linke obere Ecke.

```
   #define CLEAR              printf("\033[2J")
```

2. Erase Line

Sequenz: `ESC[K` Mnemonik: `EL` (Erase Line)

Die Zeile, in der sich der Cursor aktuell befindet, wird von einschließlich der Cursorposition bis zum Zeilenende gelöscht.

```
#define CLEAR_LINE          printf("\033[K")
```

Grafikmodusfunktionen

Mit Hilfe der folgenden ANSI-Control-Sequenzen kann die Bildschirmdarstellung verändert werden. Da einige dieser Sequenzen für DOS definiert wurden, funktionieren unter Umständen nicht alle Sequenzen auch in DOS-Fenstern bzw. in Konsolen.

1. Set Graphics Rendition

Sequenz: `ESC[P;...;Pm` Mnemonik: `SGR` (Set Graphics Rendition)

Die `SGR`-Sequenz gestattet das Einstellen spezieller grafischer Darstellungen. Die Art der Funktion wird durch den Wert des Parameters `P` bestimmt. Die folgende Liste zeigt möglichen Werte, die der Parameter `P` annehmen kann.

Parameter P	**Wirkung**
0	Alle Attribute werden zurückgesetzt.
1	Verstärkte Intensität (Fett) einschalten.
4	Unterstreichen einschalten (nur Monochrom-Bildschirme).
5	Blinken einschalten.
7	Inversdarstellung einschalten.
8	Unsichtbare (versteckte) Ausgabe.
30	Vordergrundfarbe schwarz.
31	Vordergrundfarbe rot.
32	Vordergrundfarbe grün.
33	Vordergrundfarbe gelb.
34	Vordergrundfarbe blau.
35	Vordergrundfarbe violett.
36	Vordergrundfarbe kobaltblau.
37	Vordergrundfarbe weiß.
40	Hintergrundfarbe schwarz.
41	Hintergrundfarbe rot.
42	Hintergrundfarbe grün.

43	Hintergrundfarbe gelb.
44	Hintergrundfarbe blau.
45	Hintergrundfarbe violett.
46	Hintergrundfarbe kobaltblau.
47	Hintergrundfarbe weiß.

Hier nur einige Beispiele für die Umsetzung in die entsprechenden C-Makros:

```
#define BOLD              printf("\033[1m")
#define UNDERSCORE        printf("\033[4m")
#define BLINK             printf("\033[5m")
#define INVERSE           printf("\033[7m")
#define BOLD_INVERSE      printf("\033[1;7m")
#define RED_ON_WHITE      printf("\033[31;47m")
#define ATTRIBUTE_OFF     printf("\033[0m")
```

Der Nachteil der letzten Sequenz besteht darin, dass damit **alle** gerade aktiven Attribute zurückgesetzt werden. Hier müssten parallel noch Informationen über die gerade gesetzten Attribute verwaltet werden (z. B. in einem Bitfeld). Möchte man nun ein Attribute zurücksetzen, so muss man dazu mit dem Makroaufruf `ATTRIBUTE_OFF` alle zurückgesetzt werden und anschließend die restlichen nacheinander wieder explizit aktivieren.

2. Set Mode

Sequenz: `ESC[=Ph` Mnemonik: `SM` (Set Mode)

Sowohl die Bildschirmbreite als auch der -typ können mit der `SM`-Sequenz eingestellt werden. Der Parameter `P` kann folgende Werte (für CGA-Bildschirme; für VGA-Bildschirme usw. gibt es weitere Werte) annehmen:

Parameter P	**Wirkung**
0	Textbildschirm: 25 Zeilen * 40 Spalten (schwarz/weiß).
1	Textbildschirm: 25 Zeilen * 40 Spalten (Farbe).
2	Textbildschirm: 25 Zeilen * 80 Spalten (schwarz/weiß).
3	Textbildschirm: 25 Zeilen * 80 Spalten (Farbe).
4	Grafikbildschirm: 320 * 200 Punkte (Farbe).
5	Grafikbildschirm: 320 * 200 Punkte (schwarz/weiß).
6	Grafikbildschirm: 640 * 200 Punkte (schwarz/weiß).

7	Wrap-Modus einschalten. Sobald das Ende einer Zeile erreicht ist, erfolgt automatisch ein Zeilenvorschub. Das bedeutet, dass Zeichen, die über die aktuelle Zeile hinausgehen, in die nächste Zeile übernommen werden.

Hier wieder nur einige Beispiele für die Umsetzung in die entsprechenden C-Makros:

```
#define TEXT_COLOR_25_80   printf("\033[=3h")
#define WRAP_MOD_ON        printf("\033[=7h")
```

3. Reset Mode

Sequenz: `ESC[=Pl` Mnemonik: `RM` (Reset Mode)

Die `RM`-Sequenz setzt die mit der `SM`-Sequenz gesetzten Attribute zurück. Die Parameterwerte für den Parameter `P` sind die gleichen wie bei der zurvor beschriebenen `SM`-Sequenz.

```
#define WRAP_MOD_OFF       printf("\033[=7l")
```

Änderung der Tastaturbelegung

Folgende Control-Sequenz erlaubt eine Veränderung der Tastaturbelegung:

Sequenz: `ESC[#;...;#p`

Zu beachten ist, dass das Symbol # sowohl einen numerischen Parameter (also eine Dezimalzahl) als auch eine Zeichenfolge (von Anführungszeichen eingeschlossen) repräsentieren kann. Diese Sequenz ist in keine ISO- oder ANSI-Norm zu finden, entspricht jedoch der Vorschrift zur Bildung einer ANSI-Control-Sequenz. Das abschließende Zeichen `p` spezifiziert diese Sequenz als eine private ANSI-Control-Sequenz.

Der erste Parameter # ist der Tastencode als dezimaler Wert. Handelt es sich hierbei um eine Funktions- (F1 bis F12) oder Sondertaste (z.B. Cursor- bzw. Pfeiltasten, Ins (Einfg), Del (Entf), ...), die durch zwei ASCII-Zeichen gekennzeichnet ist, ist der erste Parameter eine 0 und als nächster Parameter folgt der Tastencode. Alle weiteren Parameter (Dezimalzahlen und/oder Zeichenketten) der Sequenz stellen die Zeichenfolge dar, die nach dem Drücken der durch den ersten bzw. den beiden ersten Parametern beschriebenen Taste erzeugt werden sollen. Hierzu nun einige Beispiele:

Die folgenden Sequenzen verändern die Belegung der Tasten 'Z', 'z', 'Y' und 'y'. Das bedeutet, dass nach dem Drücken der Taste 'Z' bzw. 'z' das Zeichen 'Y' bzw. 'y' erscheint (und umgekehrt).

```
printf("\033[122;121p")   /* 'z' wird zu 'y' */
printf("\033[90;89p")     /* 'Z' wird zu 'Y' */
printf("\033[121;122p")   /* 'y' wird zu 'z' */
```

```
printf("\033[89;90p")     /* 'Y' wird zu 'Z' */
```

Die nächste Sequenz legt auf die Funktionstaste F8 den DOS-Befehl `dir`, abgeschlossen mit einem `RETURN` (ASCII-Wert 13), damit die sofortige Ausfüung des Befehls veranlasst wird. Der Tastencode für die Funktionstaste F8 lautet `0;66`.

```
printf("\033[0;66;\"dir\";13p")
```

Hier werden noch einmal alle Escape-Sequenzen zusammengestellt und als Headerdatei aufgelistet:

 *escapesequenzen.h*

```
#ifndef escapesequenzen_h
   #define escapesequenzen_h escapesequenzen_h

   #define POSITION(Ze, Sp)     printf("\033[%d;%dH",Ze,Sp)
   #define HOME                 printf("\033[H")
   #define UP(Anz)              printf("\033[%dA",Anz)
   #define UP_LINE              printf("\033[A")
   #define DOWN(Anz)            printf("\033[%dB",Anz)
   #define DOWN_LINE            printf("\033[B")
   #define RIGHT(Anz)           printf("\033[%dC",Anz)
   #define ONE_POS_RIGHT        printf("\033[C")
   #define LEFT(Anz)            printf("\033[%dD",Anz)
   #define ONE_POS_LEFT         printf("\033[D")

   #define STORE_POS            printf("\033[s")
   #define RESTORE_POS          printf("\033[u")
   #define ACT_POS              printf("\033[6n")

   #define CLEAR                printf("\033[2J")
   #define CLEAR_LINE           printf("\033[K")

   #define ATTRIBUTE_OFF        printf("\033[0m")
   #define BOLD                 printf("\033[1m")
   #define UNDERSCORE           printf("\033[4m")
   #define BLINK                printf("\033[5m")
   #define INVERSE              printf("\033[7m")
   #define INVISIBLE            printf("\033[8m")

   #define FORECOLOR_BLACK      printf("\033[30m")
   #define FORECOLOR_RED        printf("\033[31m")
   #define FORECOLOR_GREEN      printf("\033[32m")
```

```
    #define FORECOLOR_YELLOW     printf("\033[33m")
    #define FORECOLOR_BLUE       printf("\033[34m")
    #define FORECOLOR_VIOLETT    printf("\033[35m")
    #define FORECOLOR_KOBALTBLUE printf("\033[36m")
    #define FORECOLOR_WHITE      printf("\033[37m")
    #define BACKCOLOR_BLACK      printf("\033[40m")
    #define BACKCOLOR_RED        printf("\033[41m")
    #define BACKCOLOR_GREEN      printf("\033[42m")
    #define BACKCOLOR_YELLOW     printf("\033[43m")
    #define BACKCOLOR_BLUE       printf("\033[44m")
    #define BACKCOLOR_VIOLETT    printf("\033[45m")
    #define BACKCOLOR_KOBALTBLUE printf("\033[46m")
    #define BACKCOLOR_WHITE      printf("\033[47m")

    #define TEXT_BW_25_40        printf("\033[=0h")
    #define TEXT_COLOR_25_40     printf("\033[=1h")
    #define TEXT_BW_25_80        printf("\033[=2h")
    #define TEXT_COLOR_25_80     printf("\033[=3h")
    #define GRAFIC_COLOR_320_200 printf("\033[=4h")
    #define GRAFIC_BW_320_200    printf("\033[=5h")
    #define GRAFIC_BW_640_200    printf("\033[=6h")
    #define WRAP_MODE_ON         printf("\033[=7h")
    #define WRAP_MODE_OFF        printf("\033[=7l")
#endif
```