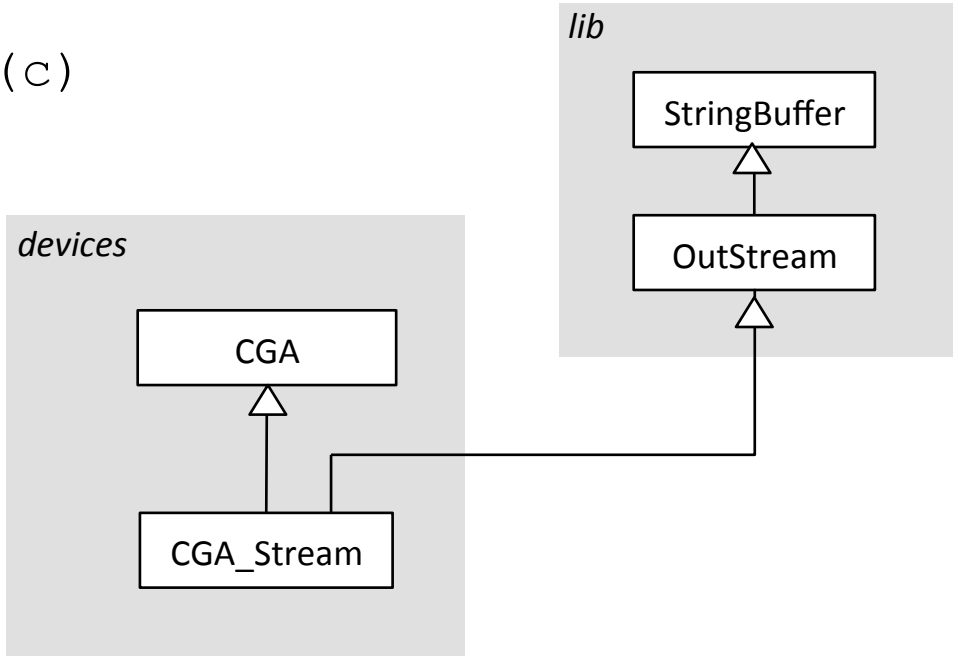
CGA-Programmierung

Ausgabestrom

- `StringBuffer: put(c), flush()`
  - Sinn der Pufferung? → Performance
  - Sinnvolle Puffergröße? → eine Zeile (80 Zeichen)
- `OutStream`: ähnlich C++ std::ostream (erweitert StringBuffer)
  - Formatierung, Zahlendarstellung
  - verwendet `StringBuffer::put(c)`
- `CGA_Stream`:
  - implementiert `flush()`
    -> Ausgabe auf Screen
    mithilfe von CGA
- `CGA`: low-level Zugriff auf Screen

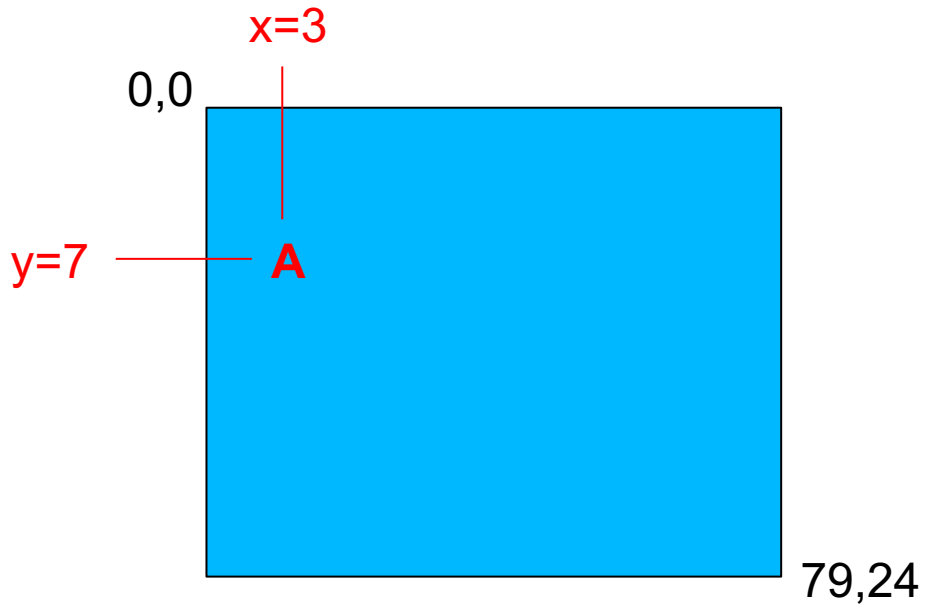


CGA::print

- `print(char* string, int n, unsigned char attrib)`
- Wird von von `flush()` in `CGA_Stream` gerufen
- Gibt die Zeichen im Buffer an der aktuellen Cursor-Position aus
  - Verwendet für die Ausgabe die Funktion `show(x,y,c,attrib)`
  - Die Text-Cursor-Position wird in der Hardware verwaltet und mithilfe von `getpos` und `setpos` gelesen beziehungsweise geschrieben

Text-Cursor

- Die Position ist ein 16 Bit Offset zur linken oberen Ecke
- Die Textauflösung ist 80x25 Zeichen
- Cursor-Position im Beispiel = (3,7)
- Offset = 3*80 + 7 = 247



Text-Cursor

- Der 16 Bit Offset wird in folgende Index-Register geschrieben / gelesen
- Jedes Indexregister kann nur ein Byte schreiben / lesen
- Die Auswahl eines Indexregisters 14 oder 15 erfolgt über Port 0x3d4
- Die Bytes werden über das Datenregister geschrieben / gelesen

Port	Register	Zugriffsart
3d4	Indexregister	nur schreiben
3d5	Datenregister	lesen und schreiben

Index	Register	Bedeutung
14	Cursor (high)	Zeichenoffset der Cursorposition
15	Cursor (low)	

Anzeige von Zeichen

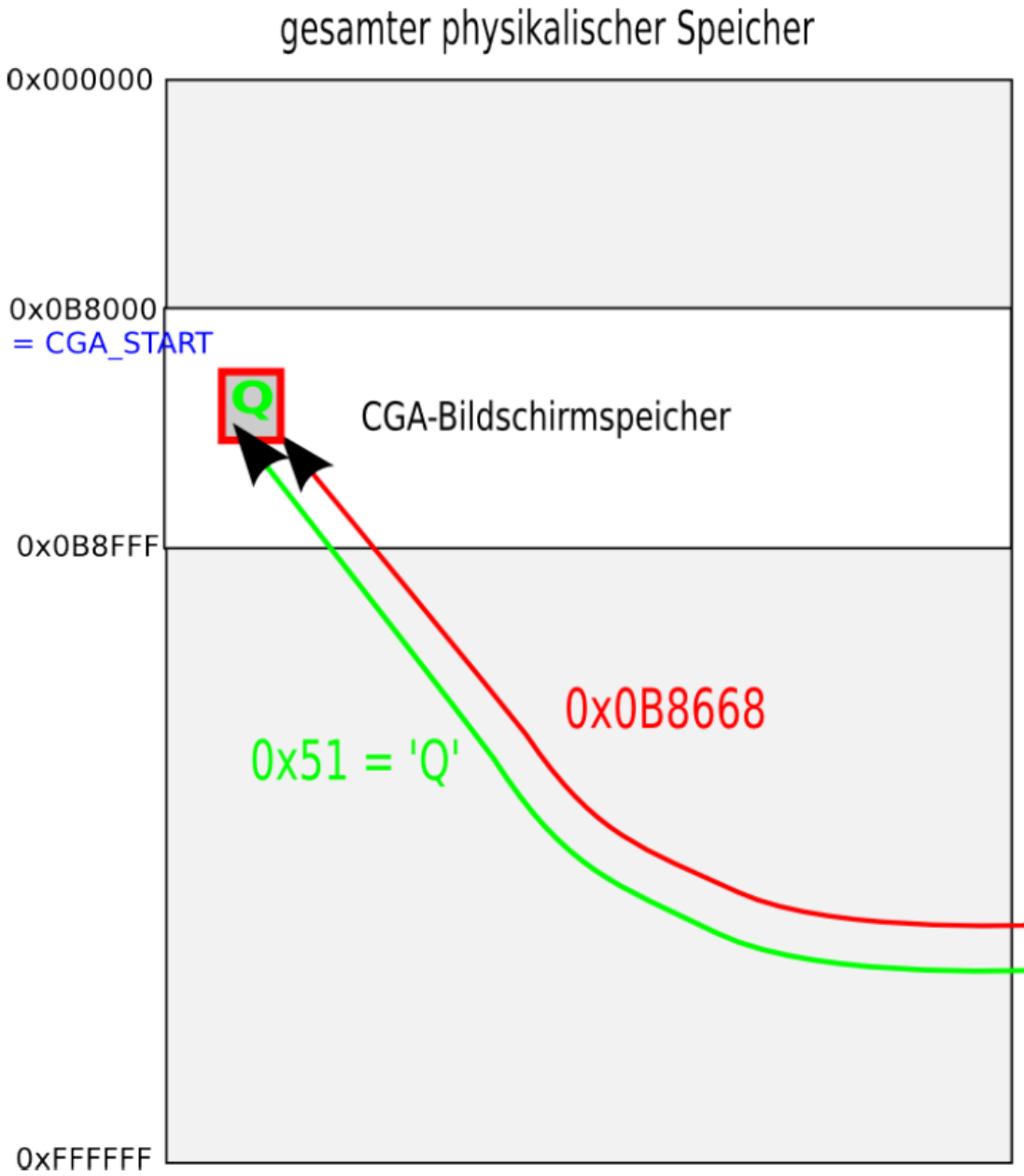
- Wird von von `print` gerufen
- `show(x,y,c,attrib)`
  - Zeichen `c` mit Attribut `attrib` an Position `x, y`
  - Code aus dem C++-Crashkurs:

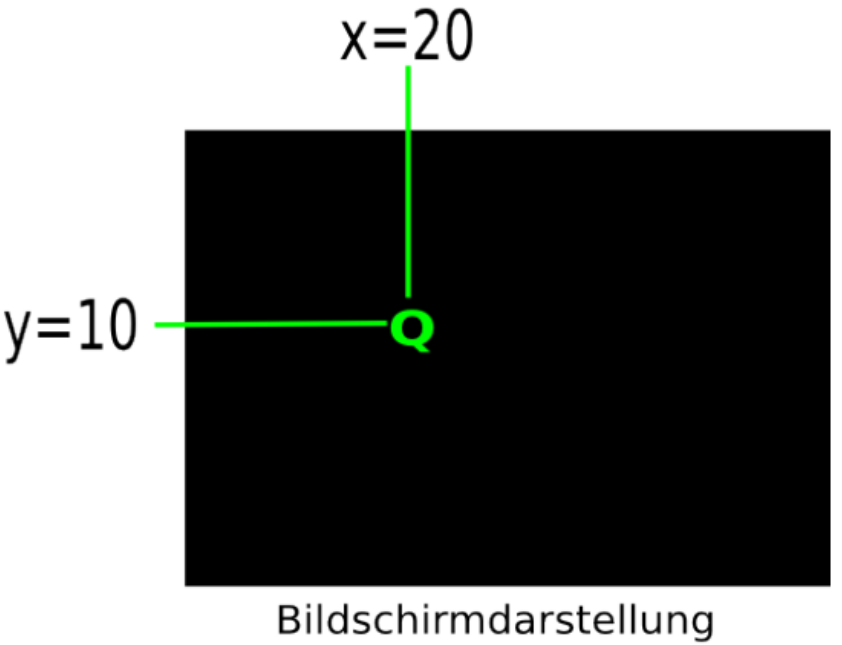
```
char *CGA_START = (char *)0xb8000;
char *pos;
int x=20, y=20;

pos = CGA_START + 2*(x + y*80);
*pos = 'Q';
```

  - Fehlt hier noch etwas?

Anzeige von Zeichen






```
#define CGA_START 0xb8000
char *pos;
x=20; y=10;
pos = (char *)CGA_START + 2*(x + y*80);
*pos = 'Q';
```

Anzeige von Zeichen

- Je zwei Bytes im Bildspeicher pro Bildposition!
- Gerade Adressen: ASCII-Code
- Ungerade Adressen: Attributbyte

```
char *CGA_START = (char *)0xb8000;
char *pos;
int x=20, y=20;

pos = CGA_START + 2*(x + y*80);
*pos = 'Q';
*(pos + 1) = 0x0f; // weiss auf schwarz
```

Attribut-Byte

- Zu jedem Zeichen können die Merkmale Vordergrundfarbe, Hintergrundfarbe und Blinken einzeln festgelegt werden.
- Für diese Attribute steht pro Zeichen ein Byte zur Verfügung,

Darstellungsattribute	
Bits 0-3	Vordergrundfarbe
Bits 4-6	Hintergrundfarbe
Bit 7	Blinken

Attribut-Byte

- Im CGA-Textmodus stehen die folgenden 16 Farben zur Verfügung:

Farbpalette			
0	Schwarz	8	Dunkelgrau
1	Blau	9	Hellblau
2	Grün	10	Hellgrün
3	Cyan	11	Hellcyan
4	Rot	12	Hellrot
5	Magenta	13	Hellmagenta
6	Braun	14	Gelb
7	Hellgrau	15	Weiß

- Da für die Hintergrundfarbe im Attributbyte nur drei Bits zur Verfügung stehen, können auch nur die ersten acht Farben zur Hintergrundfarbe gewählt werden.

Weiterführende Informationen

- Wer mehr zum Thema VGA-Grafikkarten-Programmierung lesen möchte, sei auf das FreeVGA-Projekt verwiesen:

  http://www.osdever.net/FreeVGA/home.htm

- Ist nicht notwendig für unsere Aufgabe!