1 Aufgabe: Statische 2D-Arrays

Als Erweiterung der Tutoriumsaufgabe möchten wir uns vornehmen, nicht nur einen Regenbogen zu malen, sondern den ganzen Bildschirm voll mit bunten Kästchen. Problem: Der Bildschirm ist zweidimensional. Also können wir ihn am besten mit einem zweidimensionalen Array abbilden. Das können wir beispielsweise folgendermaßen deklarieren:

```
int screen[4][3];
```

Unser kleiner Bildschirm (ihr könnt euren natürlich gerne größer machen) hat nun die Breite vier und die Höhe drei Pixel. Welche der Dimensionen unseres Arrays wir dabei als Breite bzw. Höhe interpretieren, ist dabei uns selbst überlassen. Wir müssen nur aufpassen, die verschachtelte **for**-Schleife zum Traversieren des Bildschirms entsprechend aufzubauen.

Als nächstes definieren wir uns eine Funktion, die uns eine zufällige Farbe generiert:

```
#include <stdlib.h> /* for random() */

static int randomColor(Rainbow rainbow){
  return rainbow.color[random()%rainbow.length];
}
```

und initialisieren unseren Bildschirm mit lustigen bunten Zufallsfarben.

```
for(int y= 0; y<sizeof(screen[0])/sizeof(int); y++){
  for(int x= 0; x<(sizeof(screen)/sizeof(screen[0])); x++){
    screen[x][y]= randomColor(rainbow);
  }
}
```

Betrachtet genau die Abbruchbedingungen und überlegt euch, weshalb das so funktionieren muss.

1.1 Aufgabe: Programmierung der Ausgabeschleife

Zur Übung schreibt euch nun die zugehörige Ausgabe selbst! Als Template könnt ihr euren **rainbow** Code aus dem Tutorium benutzen.

Lösung siehe **screen1.c**:

1.2 Speicherlayout mehrdimensionaler Arrays

Wie sieht nun aber das Ganze im Speicher aus? Denn der ist ja nun mal eindimensional. Angenommen, unser Programm produziert folgende Ausgabe:

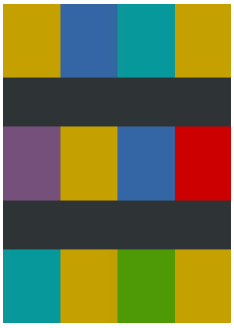

Figure 1: Die Ausgabe eures Programms

Dann liegen die Spalten unseres Arrays (mit fiktiven Adressen) hintereinanderweg im Speicher:

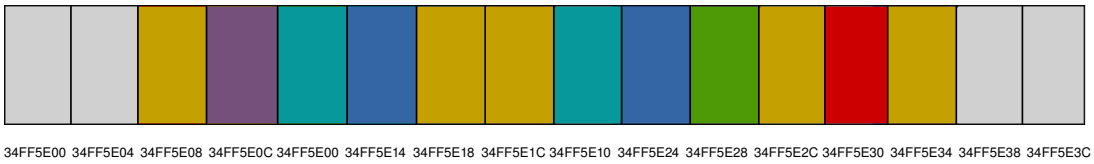

Figure 2: Speicherlayout mehrdimensionaler Arrays

Die Indizierung über **[][]** wird praktischerweise zur Compiletime in einen eindimensionalen Index umgerechnet.

Das bedeutet insbesondere aber, dass bei einem mehrdimensionalen Array als Funktionsargument in allen ausser der linkesten eckigen Klammer die Längen *explizit* angegeben werden müssen! Der Compiler kann sonst keinen Code für die Indizierung generieren!

1.3 Aufgabe: Mehrdimensionale Arrays als Funktionsargumente

Deklariere und definiere zwei Funktionen **initScreen** und **printScreen**, und benutze sie statt der for-Schleifen, um deinen **screen** zu initialisieren und auszugeben!

Lösung siehe **screen2.c**:

1.4 Aufgabe: Mehrdimensionale Arrays ``objektorientiert''

Hartgecoded die Arraylängen im ganzen Programm zu verstreuen ist hässlich, hässlich, hässlich (und absolut unwartbar)! Ich überlasse es daher euch als Übung, auch den Code für 2D-Arrays objektorientiert umzuschreiben. Eure Bildschirmgröße könnt ihr dabei als zur Compilezeit bekannt annehmen. Wie man das Problem auf etwas flexiblere Weise lösen kann, werdet ihr heute Nachmittag sehen.