Übungsblatt 2

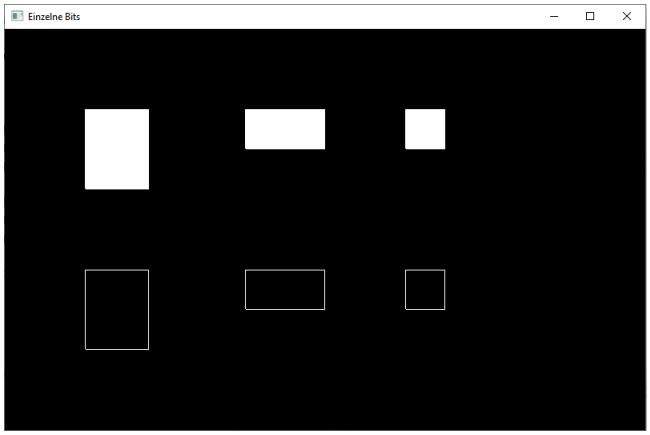
In einigen der Aufgaben dieses Übungsblatts sollen wieder Grafiken erzeugt werden. Benutzen Sie dazu das auf dem letzten Übungsblatt vorgestellte Framework!

1. Schreiben Sie eine Klasse `BinaryVisuals`, die die statischen Methoden mit den folgenden Deklarationen umsetzt:

   (a) `static void prepareBit(ViewPortGL& vp, int xk, int yk, int width, int height, bool isOne)`

   Diese Methode dient zur (Vorbereitung der) Darstellung eines einzigen Bits in einem ViewPort. Dabei soll ein Bit als Rechteck dargestellt werden, das genau dann ausgefüllt wird, wenn es auf 1 gesetzt ist (wird durch Parameter `isOne` übergeben).

   Die Methode zeichnet das Rechteck so, dass sein linkes, oberes Eck genau auf den Koordinaten `(xk, yk)` liegt. Die Darstellung soll die Breite `width` und die Höhe `height` besitzen.

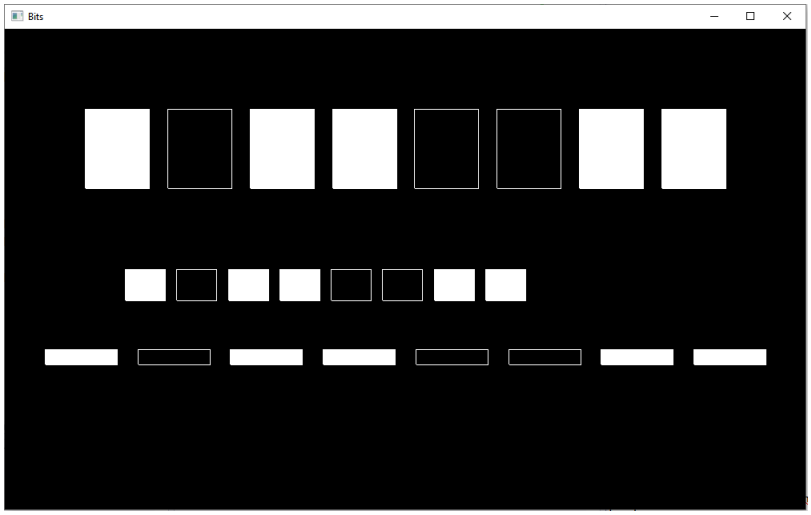
   

   Das Bild zeigt in der oberen Reihe die Darstellungen von 1-Bits unterschiedlicher Größen und in der unteren Reihe die Darstellungen von 0-Bits.

   (b) `static void prepareRepresentation(ViewPortGL& vp, unsigned char value, int xk, int yk, int width, int height)`
   Diese Methode dient dazu, einen Wert des Datentyps `unsigned char` darzustellen. Die Bits des Wertes sollen waagerecht nebeneinander angezeigt werden. Das niedrigwertigste Bits sitzt dabei ganz rechts, das höchstwertigste ganz links. Die Position und Größe der Darstellung wird (im selben Sinn wie in der Bit-Aufgabe) durch die Parameter `xk, yk, width, height` vorgegeben.

   Achten Sie bei Ihrer Implementation darauf, dass der durch diese Parameter vorgegebene Raum möglichst optimal und ohne Ränder ausgenutzt wird. Damit man die Bits gut unterscheiden kann, sollte zwischen je zwei Bits ein angemessener Zwischenraum gelassen werden.

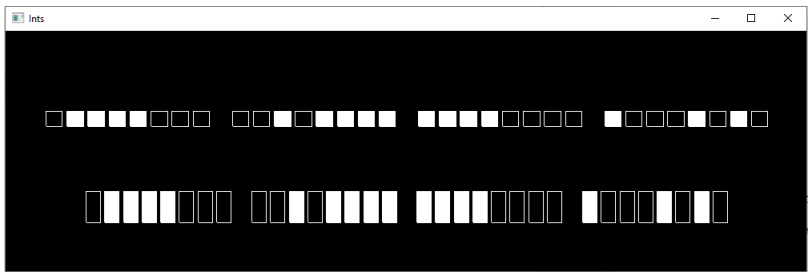
   Achten Sie außerdem darauf, dass ein `unsigned char` nicht zwangsläufig 8 Bits besitzen muss (obwohl das fast immer der Fall ist). Auf die tatsächliche Bitbreite von `char`s kann über die Bibliothek `limits.h` und das dort definierte Makro `CHAR_BIT` zugegriffen werden.



Darstellung des Wertes `179` als 8-Bit `unsigned char` in verschiedenen Größen.

(c) `static void prepareRepresentation(ViewPortGL& vp, unsigned int value, int xk, int yk, int width, int height)`
Diese Methode soll analog zur obigen arbeiten, nun aber Werte vom Typ `unsigned int` anzeigen. Benutzen Sie zu ihrer Umsetzung die Methode für `unsigned chars`! Benutzen Sie entsprechende Bitoperationen, um die `chars` aus dem `int` herauszufiltern.

Beachten Sie auch hier, dass der Datentyp `unsigned int` nicht auf jedem Rechner gleich viele Bits besitzt (aber immer ein Vielfaches der Anzahl von Bits in einem `char`). Erinnern Sie sich, dass die Anzahl der Bytes (Chars) von einem Datentyp über die Funktion `sizeof` erfragt werden kann.



Darstellung des Wertes `2016407690` als 32-Bit `unsigned int` in verschiedenen Größen.

(d) (freiwillig)
Erstellen Sie eine Animation, in der Dargestellt wird, wie ein `int`-Wert von 0 bis zu seinem Maximalwert binär hochgezählt wird.

2. Schreiben Sie eine Funktion mit der Deklaration

```
unsigned int exchangeHalves(unsigned int value, int fromBit, int toBit),
```

die zwei Bitblöcke in dem gegebenen Wert `value` wie folgt gegeneinander austauscht.

Zunächst wird durch die Parameter `fromBit` und `toBit` eine Region von Bits in `value` abgegrenzt. Die Funktion soll nur auf diesem Bitbereich arbeiten und alle anderen Bits unangetastet lassen.

Beispiel: Ist der 32-Bit Wert `1000 1110 0000 1111 1010 1110 1001 1001` gegeben, (das ist die Binärdarstellung von 2383392409), ist `fromBit = 7` und `toBit = 12`, so ist der unten farblich markierte Bereich ausgewählt:

Index	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bits	1	0	0	0	1	1	1	0	0	0	0	0	1	1	1	1	1	0	1	0	1	1	1	0	1	0	0	1	1	0	0	1

Die Funktion muss nur richtig arbeiten, wenn in diesem Bereich geradzahlig viele Bits liegen. Ihre Aufgabe besteht darin, die linke Hälfte dieses Bereichs gegen seine rechte auszutauschen.

Beispiel:
Vor dem Umtausch:

Index	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bits	1	0	0	0	1	1	1	0	0	0	0	0	1	1	1	1	1	0	1	0	1	1	1	0	1	0	0	1	1	0	0	1

Nach dem Umtausch:

Index	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bits	1	0	0	0	1	1	1	0	0	0	0	0	1	1	1	1	1	0	1	1	0	1	0	1	1	0	0	1	1	0	0	1

Visualisieren Sie, nachdem Sie die Funktion implementiert haben, dass sie funktioniert!

3. Schreiben Sie eine Funktion mit der Deklaration

```
unsigned int reverse(unsigned int b),
```

die einen Wert zurückgibt, in dessen Binärdarstellung die Bits gerade anders herum auftauchen als im gegebenen `b`.

Genauer: Ist $w = a_1 a_2 \dots a_n$ ein binäres Wort, so sei $w^R \stackrel{\text{def}}{=} a_n a_{n-1} \dots a_2 a_1$ das selbe Wort, nur umgedreht (alle a_i sind hier aus $\{0, 1\}$). Die obige Funktion `reverse` produziert also zum gegebenen `b` die Ausgabe $\texttt{b}^R$.

Hinweis zur Lösung der Aufgabe: Sie kann recht einfach rekursiv mit Hilfe von Aufgabe 2 gelöst werden, vermöge der folgenden Überlegungen: Für eine Zweierpotenz n (oder $n = 0$) sei wieder $w = a_1 \dots a_{n/2} a_{n/2+1} \dots a_n$ ein binäres Wort mit den Bits a_i. Wir definieren $w^H \stackrel{\text{def}}{=} a_{n/2+1} \dots a_n a_1 \dots a_{n/2}$ (analog zu Aufgabe 2). Außerdem sei

$$f(w) \stackrel{\text{def}}{=} \begin{cases} w & \text{, falls } w \text{ die Länge 0 hat} \\ f(w_1)f(w_2) & \text{in allen anderen Fällen, wobei } w^H = w_1 w_2 \text{ vorausgesetzt wird} \end{cases}$$

Beispiel zum Verständnis der Definition von f: Ist $w = 10011101$, dann ist $w^H = 11011001$ und damit ergibt sich $w_1 = 1101$ und $w_2 = 1001$. Laut Definition gilt darum $f(w) = f(w_1)f(w_2) = f(10011101) = f(1101)f(1001)$.

Es lässt sich leicht zeigen: $w^R = f(w)$

4. Nach diesen etwas theoretischeren Aufgaben gönnen wir uns noch eine etwas buntere: Schreiben Sie ein Programm, das ein Bild ähnlich dem folgenden produziert: