AIK Programmieren 1 Übung: Bit-Operationen

Klaus Kusche

Ausgabe von Zahlen binär und hexadezimal

Schreib ein Programm, das alle auf der Befehlszeile angegebenen **`int`**-Zahlen
der Reihe nach *als 32-stellige Binärzahl (mit führenden Nullen) ausgibt*.

Hinweise:

- Verwende zur Umwandlung in binär und zur Ausgabe
  weder **`printf`**-*Formate noch Multiplikation, Division oder Restbildung*
  (einlesen darfst du die **`int`**'s wie bisher mit **`atoi`**).

  Extrahiere stattdessen die *einzelnen Bits* für die Ausgabe der Reihe nach
  mit Bitoperationen aus dem **`int`**.
- Ein *einzelnes Zeichen* ***c*** kannst du mit der Funktion **`putchar(`*`c`*`)`** (aus **`stdio.h`**)
  ausgeben, ***c*** ist der ASCII-Wert des auszugebenden Zeichens, im einfachsten Fall
  eine **`char`**-Konstante wie z.B. **`'0'`**.
- *Welchen* **`int`**-*Typ* sollte die Variable für die eingelesene Zahl
  (und auch Hilfsvariablen für die Bitoperationen) haben,
  damit es bei den Bit-Schiebe-Operationen keine Überraschungen gibt?

Zusatzaufgaben:

- Kannst du die Binärzahl schön in *mit Zwischenräumen getrennten Vierergruppen*
  ausgeben?

  Auch das solltest du *ohne* Multiplikation, Division oder Restbildung schaffen!
  (welche Bit-Operationen kann eine Restrechnung mit 4 ersetzen?)
- Kannst du die eingelesene Zahl auch *als 8-stellige Hexzahl* ausgeben?
  (wieder mit führenden Nullen und aufgeteilt in zwei Vierergruppen)

  In diesem Fall musst du nicht einzelne Bits aus dem **`int`** extrahieren,
  sondern in jedem Schleifendurchlauf *vier aufeinanderfolgende Bits gemeinsam*.
- Schaffst du es, die auszugebende Ziffer **`'0'`** oder **`'1'`** bei der Binär-Ausgabe
  direkt aus der Zahl zu berechnen, *ohne* ein **`if`** oder einen Conditional Operator
  zu verwenden?

  Erinnere dich: Mit **`char`**-Werten kann man rechnen…

  Bei der Hex-Ausgabe solltest du mit einem einzigen **`if`** oder **`? :`**
  für die Unterscheidung zwischen Hex **`'0'`**-**`'9'`** und Hex **`'A'`**-**`'F'`** auskommen
  und nicht 16 einzelne Fälle unterscheiden.

  Alternativer Trick: Verwende den extrahierten Wert 0 bis 15 als *Index in ein Array*,
  das die auszugebenden Zeichen **`0123456789ABCDEF`** enthält.

LED's rotieren

Eine Gruppe AIK's hat vor Jahren als Projekt eine **LED-Matrix mit 14 * 14 LED's** gebaut. Die anzuzeigenden Muster sind in einem Array von 14 **`int`**-Werten gespeichert:

- *Jedes Array-Element* entspricht *einer Zeile* der LED-Matrix
  (Element 0 ist die oberste LED-Zeile).
- Von jedem einzelnen **`int`**-Wert sind jeweils *die hinteren 7 Bits der hinteren beiden Bytes* genutzt. Wenn man die Bits des **`int`** von 0 (*niederwertigstes* Bit, ganz rechts) bis 31 (*höchstwertiges* Bit, ganz links) nummeriert, sind das die Bits 0-6 und 8-14.

  Die restlichen Bits (7, 15-31) sind *ungenutzt* und sollten *stets 0* sein.
- Jedes genutzte Bit entspricht einer LED der jeweiligen Zeile, *von links nach rechts*:

Bit	31							24	23							16	15							8	7						0
LED	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	2	3	4	5	6	7	-	8	9	10	11	12	13	14

Ich habe ein *Hauptprogramm* geschrieben, das ein solches Muster enthält und grafisch anzeigt. Lade dir mein Programm und die SDL herunter und bringe es mit meiner SDL-Anleitung zum Laufen.

Eigentlich sollte das Programm das gespeicherte Muster **von rechts nach links laufend** anzeigen. Es ruft dazu alle 100 Millisekunden für jede Zeile die Funktion **`rotate`** auf.

Derzeit macht diese Funktion noch gar nichts, aber sie sollte das Muster in der Zeile *um eine LED nach links rotieren*:

- Die Funktion hat einen **`int`**-*Parameter*: Die LED-Daten *einer Zeile* (wie oben beschrieben).
- Der Returnwert soll wieder ein **`int`** mit den LED-Daten sein, er soll *dieselbe Zeile* des Musters *rotiert* enthalten:
  Der alte Wert von LED-Spalte 1 wandert in die LED-Spalte 14, alle anderen LED-Spalten wandern um 1 nach links.

Schreib diese Funktion **`rotate`**!

Tipp:

- Überlege dir, *welche Bits des alten Zeilen-Wertes* in *welchen Bits des Ergebnisses* landen sollen. Du wirst das Ergebnis *aus drei Teilen zusammensetzen* müssen: *Zwei Bits* der alten Zeile müssen *einzeln* ausgeschnitten und verschoben werden, die *restlichen 12 Bits* können *gemeinsam* ausgeschnitten und an die richtige Stelle gebracht werden.
- Du solltest *nur mit Bit-Operationen auskommen* (*ohne* **`if`** und ohne *Schleifen* sowie ohne Rechen-Operationen)!

Zusatzaufgabe:

Schaffst du es auch, die Zeilen *zu spiegeln statt zu rotieren*?
(d.h. die LED-Spalte 1 wird mit Spalte 14 vertauscht, 2 mit 13, 3 mit 12 usw. bis 7 mit 8)

Das ist etwas trickreich, du wirst eine *Schleife* brauchen.

Möglichkeit 1:

Man verwendet eine Schleife mit *zwei Zählvariablen*
(der *Position* des *linken* und des *rechten* zu vertauschenden Bits im **`int`**):
Man schneidet in jedem Durchlauf zuerst das rechte Bit aus, schiebt es an die Stelle des linken Bits, und fügt es zum ursprünglich leeren Ergebnis dazu.
Dann schneidet man das linke Bit aus, schiebt es an die Stelle des rechten Bits, und fügt es auch zum Ergebnis dazu.

Möglichkeit 2:

Man schiebt 15 Bits *einzeln* der Reihe *nach rechts* (*nach hinten*) aus dem alten Wert *hinaus* und *von rechts* (von hinten) in das Ergebnis *hinein*
(es schadet nichts, dabei das unbenutzte 0-Bit an Stelle 7 mitzuschieben):

Die Schleife wiederholt daher folgende Schritte:
- *Ergebnis ein Bit nach links* schieben ==› letztes Bit des Ergebnisses wird frei
- *Letztes Bit des Originalwertes* ausschneiden und als letztes Bit zum Ergebnis dazugeben
- *Originalwert ein Bit nach rechts* schieben ==› soeben kopiertes Bit fällt weg