Programmieren C: Funktionen & Arrays, `calloc`: Turmrechnen

Klaus Kusche

Eine beliebte Strafaufgabe ist das "Turmrechnen": Man muss händisch eine *große Zahl* (20 Stellen und mehr) *nacheinander mit 2, 3, … 9 multiplizieren* und anschließend *durch 2, 3, … 9 dividieren* (und es kommt am Ende hoffentlich die ursprüngliche Zahl heraus!).

Eine "direkte" Implementierung in C scheitert daran,
dass ein **`int`** (und sogar ein 64 Bit **`int`**) *zu klein* für die Darstellung solcher Zahlen ist.
Wir umgehen das, indem wir die Zahlen als *Array* von **`int`**'s darstellen, ein **`int`** *pro Ziffer*
(in der Realität würde man für größtmögliche Speicher- und Zeit-Effizienz ein **`int`**-Array benutzen, in jedem **`int`**-Element die *vollen 32 Bit nutzen* (*nicht* nur eine Ziffer speichern), und alle Rechenoperationen mit 64 Bit **`int`**'s durchführen, denn zwei 32-Bit-**`int`**-Werte kann man mit 64-Bit-Rechnungen ohne Überlauf addieren und multiplizieren).

Dein Programm wird mit der *Ausgangszahl auf der Befehlszeile* aufgerufen
und soll die ursprüngliche Zahl und *alle Zwischenergebnisse* der 8 Multiplikationen
und der 8 Divisionen (bis zurück zur ursprünglichen Zahl) ausgeben.

Hinweise:

- Strukturiere dein Programm wie folgt:
  - Eine **Funktion zur Eingabe**:
    Sie bekommt den *Eingabe-String* von der Befehlszeile als *Parameter*
    und liefert einen *Pointer auf das neu angelegte Array mit den Ziffernwerten*
    als Ergebnis. *Wie* die Zahl in dem Array gespeichert wird, steht *unten*.

    Wenn die Eingabe andere Zeichen als '0' bis '9' enthält,
    soll das Programm mit einer *Fehlermeldung* abbrechen.
  - Eine **Funktion zur Ausgabe**:
    Sie bekommt ein Array mit einer *langen Zahl* als *Parameter* übergeben
    und soll diese Zahl *ziffernweise ausgeben* (kein Returnwert).

    Führende Nullen sollen dabei als *Zwischenräume* ausgegeben werden.
  - Eine **Multiplikations-Funktion**, die eine *lange Zahl* mit einem *normalen* **`int`**
    zwischen 0 und 9 (*beides als Parameter* übergeben) multipliziert,
    und zwar *"vor Ort"* (das Ergebnis steht wieder im als Parameter übergebenen ursprünglichen Array, kein Returnwert).

    Implementiere die Multiplikation so, wie man das händisch macht:
    Ziffernweise von hinten nach vorne.
    Bei der Multiplikation zweier Ziffern kommt ein *zweistelliges Ergebnis* heraus.
    Die *Einer-Stelle* des Ergebnisses wird im *Array* gespeichert,
    die *Zehner-Stelle* als *Übertrag* zum Ergebnis der nächsten Multiplikation *addiert*.

    Passt das Ergebnis nicht in das Array (Übertrag aus der vordersten Stelle),
    soll das Programm mit einer *Fehlermeldung* abbrechen.
  - Eine **Divisions-Funktion**, die eine *lange Zahl* durch einen *normalen* **`int`**
    zwischen 0 und 9 dividiert, so wie bei der Multiplikation mit dem Ergebnis wieder in der als Parameter übergebenen zu dividierenden Zahl.

Rechne wie bei der händischen Division *ziffernweise von vorne nach hinten*:
- Der *Rest* der vorigen Division wird *mit 10 multipliziert*,
  die *aktuelle Stelle* wird *dazugezählt*.
- Die entstehende zweistellige Zahl wird *dividiert*.
- Das *Divisionsergebnis* kommt ins *Array*,
  der *Divisionsrest* ist der *Übertrag* zur nächsten Division.

Bleibt bei der Division der Einer-Stelle ein *Rest*,
soll das Programm mit einer *Fehlermeldung* abbrechen.

- Für das **Array**, das die Zahl darstellt, gilt folgendes:
  - Im *ersten Element* steht, *wie lang* das Array insgesamt ist (damit man die Länge nicht bei jeder Funktion als zusätzlichen Parameter übergeben muss), und zwar in Elementen, nicht in Bytes.
  - Die *restlichen Elemente* des Arrays enthalten die aktuelle Zahl *ziffernweise, rechtsbündig* und *mit führenden Nullen* (d.h. das *letzte* Array-Element ist die *Einer-Stelle*, jedes Element enthält einen Wert zwischen 0 und 9).
  - Das Array wird in der Eingabe-Funktion *dynamisch angelegt* (mit `malloc` oder `calloc`), und zwar *um 9 Elemente größer*, als die Eingabe Ziffern hat:
    - Das *erste* zusätzliche Element ist das oben erwähnte *Längen-Element*.
    - Die *restlichen 8 Elemente* vor den eingegebenen Ziffern werden *mit 0 gefüllt*. Dadurch kann die Zahl bei den Multiplikationen mit 2…9 *wachsen*, ohne dass das Array überläuft bzw. ohne dass man es größer machen muss.

  Prüfe, ob das dynamische Anlegen des Arrays funktioniert hat,
  und *gib* das Array am Ende des Hauptprogramms wieder *frei*.