Programmieren 1 Übung: Array-Parameter, Strings, ...

Klaus Kusche

Mastermind (nur für Fortgeschrittene!)

Wir wollen dem Computer Mastermind beibringen:

- Der Benutzer denkt sich eine *beliebige Kombination* von **n** *Ziffern* zwischen **1** und **k** aus (die *Anzahl der Stellen* **n** und die *Anzahl der Möglichkeiten* pro Stelle **k** sollen als **#define**-Konstanten in deinem Programm vorgegeben sein).
  Auch mehrfach vorkommende Ziffern sind erlaubt.
- Der Computer rät irgendeine **n**-stellige Ziffernkombination und zeigt sie an.
- Der Benutzer gibt ein, wie gut die angezeigte Ziffernkombination zu seiner geheimen Zahl passt:
  - Wie viele Ziffern sind richtig *und* stehen auch schon an der richtigen Stelle?
  - Wie viele Ziffern (*außer* denen, die schon an der richtigen Stelle stehen!) kommen zwar auch in der geheimen Zahl vor, aber an *anderer* Stelle?
- Der Computer macht einen neuen Versuch, der Benutzer bewertet ihn wieder, usw., bis der Computer die geheime Zahl des Benutzers herausgefunden hat (d.h. alle Ziffern richtig und an der richtigen Stelle).
- Ziel ist, dafür möglichst wenig Versuche zu brauchen.

Die Lösungsidee ist folgende:

- Jede *Ziffernkombination* wird als *String der Länge* **n** *dargestellt* (**"1111"**, **"1112"**, ...).
  Achtung: Wie lang musst du die Strings dafür deklarieren?
- Beim Start des Programmes werden intern zuerst einmal *alle möglichen Ziffernkombinationen erzeugt* und in einem *zweidimensionalen* **char**-*Array* gespeichert (dafür schreiben wir uns eine eigene *Funktion* **init**, siehe unten).

  Das Array muss dazu **k** *hoch* **n** *Zeilen* haben, jede Zeile enthält einen Ziffernkombinations-String. Die *Zeilenanzahl* wirst du mit **pow** ausrechnen müssen, aber die *Spaltenanzahl ist fix* (eine **#define**-Konstante),
  also kannst du ein *echtes zweidimensionales Array* nehmen.
  Deklariere das Array lokal in **main** und übergib es allen Funktionen als *Parameter* (als *zweiten Parameter* solltest du immer die *Anzahl* der Zeilen übergeben)!
- Dann passiert in einer Schleife Folgendes (zähl die *Versuche* mit!):
  - Es wird eine zufällige, noch in Frage kommende Kombination aus dem Array *gewählt* (das ist eine eigene *Funktion* **waehle**, s. u.), in einen lokalen String im **main** *kopiert* (Stringfunktion!), und als Lösungsvorschlag angezeigt.
  - Dann muss der Benutzer die Anzahl der ganz richtigen und die Anzahl der halb richtigen Ziffern in diesem Lösungsvorschlag eingeben (**scanf**)
    (prüfe die Eingaben auf >= 0 und auf in Summe <= Anzahl der Ziffern!).
  - Wenn *alle Ziffern ganz richtig* sind, sind wir fertig:
    Gib die Anzahl der benötigten Versuche aus!

- Sonst werden all jene Kombinationen aus dem Array aller möglichen Kombinationen *gestrichen*, die nicht mehr in Frage kommen, weil sie nicht zum Lösungsvorschlag und seiner Bewertung passen (das wird eine *Funktion* **`streiche`**, siehe unten).

  Das Streichen geschieht, indem wir diese Ziffernkombination in unserem zweidimensionalen Array durch den *Leerstring* ersetzen, d.h. das vorderste Ziffernzeichen mit **`'\0'`** überschreiben.

Für diese Lösungsidee brauchen wir außer **`main`** vier Funktionen:

- **`init`** initialisiert das Array mit *allen Lösungsmöglichkeiten*.
  Die Funktion bekommt das zweidimensionale Array und seine Zeilenanzahl als Parameter übergeben und hat keinen Returnwert.

  Als erstes füllen wir die *erste Zeile* mit lauter **`'1'`**
  (was darfst du hinter den **`'1'`** nicht vergessen?).

  Dann füllen wir in einer Schleife die restlichen Zeilen:

  - Wir initialisieren jede Zeile zuerst einmal mit einer *Kopie* der vorigen Zeile (Stringfunktion!).
  - Dann gehen wir die Stellen der Zeile von vorn nach hinten durch, solange sie die *höchstmögliche* Ziffer enthalten, und *ersetzen* die höchstmögliche Ziffer durch **`'1'`**.
  - Die erste Stelle der Zeile, die *nicht* die höchstmögliche Ziffer enthält, wird um *eins erhöht*.

  Wenn man das richtig macht, enthält die letzte Zeile **`n`** Mal die höchste Ziffer.

- **`waehle`** bekommt ebenfalls das Array und dessen Zeilenanzahl als Parameter übergeben und liefert einen Pointer auf eine *zufällige, noch nicht gestrichene Zeile* des Arrays als Returnwert.

  Wir beginnen dafür bei einer *zufälligen* Zeile des Arrays und rücken so lange um eine Zeile weiter, bis wir eine gefunden haben, die noch nicht gestrichen ist (d.h. deren erstes Zeichen nicht die Ende-Markierung **`'\0'`** ist).

  - Wenn wir eine solche *noch nicht gestrichene Zeile* finden, geben wir einen Pointer darauf als Returnwert zurück.

    Tipp: Wenn man bei einem zweidimensionalen Array **`a`** nur eine Dimension in **`[]`** dahinterschreibt (z.B. **`a[i]`**), so bekommt man einen *Pointer* auf ein eindimensionales Array, nämlich auf *die* **`i`**-*te Zeile* von **`a`**.
  - Wenn wir beim Suchen das *Ende des Arrays* erreicht haben, fangen wir wieder beim ersten Element an.
  - Wenn wir einmal komplett im Kreis gelaufen und wieder bei der *ursprünglichen Zeile* angekommen sind, sind *alle* Kombinationen im Array schon *gestrichen*. Das passiert, wenn der Benutzer falsche Lösungs-Bewertungen eingibt: Wir beenden das Programm mit einer *Fehlermeldung*.

  Sorge in **`main`** dafür, dass bei jedem Programmlauf *andere* Zeilen zufällig gewählt werden!

- **`passt`** prüft, ob eine *beliebige Kombination* zu einem *Lösungsvorschlag und seiner Bewertung passen kann* oder nicht. Es ist die aufwendigste Funktion.

  `passt` hat *vier Parameter* (die zu prüfende Kombination als String, den Lösungsvorschlag als String, die eingegebene Anzahl der ganz Richtigen und die der halb Richtigen) und liefert ***"wahr"*** oder ***"falsch"*** als Ergebnis.

  Zum Ermitteln der halb Richtigen brauchen wir zwei **`int`**-Arrays mit so vielen Elementen, wie es Möglichkeiten für die Ziffern gibt (d.h. Array-Element 0 gehört zu Ziffer **`'1'`**, Element 1 zu **`'2'`** usw.: Wie kommst du vom *ASCII-Wert zum Index*?). Ein Array *zählt*, *wie oft jede Ziffer* in der zu prüfenden Kombination *vorkommt*, und das andere zählt, wie oft jede Ziffer im Lösungsvorschlag vorkommt. Dabei werden nur Ziffern gezählt, die *nicht* ohnehin schon *ganz richtig* sind.

  Zuerst einmal setzen wir alle Elemente beider Arrays auf 0.

  Dann machen wir eine Schleife, die *alle Stellen* von zu prüfender Kombination und Lösungsvorschlag der Reihe nach einzeln durchgeht:

  - Ist die Ziffer an dieser Stelle in der zu prüfenden Kombination und im Lösungsvorschlag gleich, *erhöhen* wir die Anzahl der *ganz Richtigen* um Eins.
  - Sonst *zählen* wir die beiden Ziffern an dieser Stelle in unseren beiden *Arrays*.

  Nach der Schleife prüfen wir, ob die *gezählte Anzahl der ganz Richtigen* der *eingegebenen Anzahl* entspricht. Wenn nein, können wir schon "falsch" zurückgeben, wenn ja, müssen wir auch noch die halb Richtigen prüfen:

  Jede Ziffer ist so oft halb richtig, wie sie *sowohl* in der zu prüfenden Kombination *als auch* im Lösungsvorschlag vorkommt (aber nicht ganz richtig war), d.h. der *kleinere* der zur Ziffer gehörenden Zähler in den beiden Arrays sagt uns, wie oft die Ziffer halb richtig war.

  Daher brauchen wir eine *zweite Schleife*, diesmal über unsere beiden *Zähler-Arrays*: Wir gehen alle Elemente der Reihe nach durch und addieren jeweils das *kleinere* der beiden Elemente zu unserer Anzahl der halb Richtigen (schreib dir eine kleine Minimum-Funktion für zwei **`int`**'s!).

  Am Ende geben wir "wahr" zurück, wenn diese Summe der eingegebenen Anzahl der halb Richtigen entspricht, und sonst "falsch".

- **`streiche`** wird mit *fünf Parametern* aufgerufen: Dem zweidimensionalen *Array* plus *Zeilenanzahl* sowie einem *Lösungsvorschlag* (als String) und seiner *Bewertung* (Anzahl der ganz Richtigen und Anzahl der halb Richtigen).
  `streiche` hat keinen Returnwert.

  Die Funktion *streicht alle nicht zum Vorschlag passenden Kombinationen* aus dem Array: **`streiche`** geht einfach in einer Schleife alle Zeilen des Arrays durch und ruft für jede Zeile, die noch *nicht* gestrichen ist, **`passt`** auf. Liefert **`passt`** "falsch", so wird die Zeile gestrichen (die erste Ziffer durch **`'\0'`** ersetzt).

  Auch hier hilft, dass ein zweidimensionales Array mit nur einem Index dahinter einen Pointer auf das eindimensionale Array der jeweiligen Zeile (d.h. einen unserer Lösungs-Strings) liefert.