Programmieren 1 Übung: Noch mehr Schleifen, `switch`, …

Klaus Kusche

1.) Würfeln

"Würfle" (d.h. berechne eine *"zufällige" Zahl* zwischen 1 und 6) immer wieder,
bis du *zwei Mal nacheinander einen Sechser* gewürfelt hast. Zähle dabei die Würfe mit.

Gib bei jedem einzelnen Wurf aus, der *wievielte Versuch* es war
und *welche Zahl* geworfen wurde.

Das Programm liest *keine Eingabe und keine Werte von der Befehlszeile*.

Achtung:

- Es soll in jedem Durchgang nur *eine* Zahl gewürfelt werden, und zwar so lange, bis in *zwei aufeinanderfolgenden Durchgängen* ein Sechser gewürfelt wurde.
  Das ist etwas anderes, als in jedem Durchgang *zwei* Zahlen auf einmal zu würfeln, bis beide gleichzeitig Sechser sind!

Tipp:

- Es gibt *zwei verschiedene Lösungsideen*:
  - Du kannst zusätzlich zur Variablen für den aktuellen Wurf noch eine zweite für den *vorigen* Wurf verwenden. Speichere dir jedesmal, bevor du einen neuen Wurf berechnest, den bisher aktuellen Wurf als vorigen Wurf! Wiederhole das, bis der aktuelle und der vorige Wurf *beide* Sechser sind (denke nach, wie man *zwei Vergleiche* in einer einzigen Schleifenbedingung macht!).

    Pass auf, dass deinem Programm keine falschen, zufälligen "Glückstreffer" schon beim ersten Wurf passieren (was ist der vorige Wurf am Anfang?)!
  - Verwende eine Variable, die die *Anzahl der aktuell aufeinanderfolgenden Sechser* zählt: Bei einem Sechser wird die Anzahl um 1 erhöht, bei jedem anderen Wurf wird die Anzahl auf 0 zurückgesetzt. Wiederhole das, bis die Anzahl 2 ist.
- Dein Programm soll bei jedem Programmlauf *andere* Zahlen würfeln.
  Erinnere dich an **`rand`** und **`srand`** (und wie man daraus eine Zahl in einem bestimmten Bereich berechnet).

Zusatzaufgabe:

- Ermittle *100 000 Mal nacheinander* so wie oben, wie viele Versuche du brauchst, bis du 2 Sechser hast, und gib die *durchschnittliche, minimale und maximale* Anzahl der Versuche aus (die Ausgabe der einzelnen Würfe und Versuche lässt du in diesem Fall besser weg!). Der Durchschnitt soll eine *Kommazahl* sein!

  Nach zwei Sechsern wird neu begonnen, d.h. ein dritter Sechser unmittelbar danach zählt *nicht* nochmals als zwei Sechser!

2.) Einfacher Taschenrechner

Schreib ein Programm, das ganz einfache Rechnungen ausrechnet:

- Die Rechnungen bestehen aus Kommazahlen und **+ − * / ^** (**^** für "hoch")[1].
- Es wird einfach Rechenzeichen für Rechenzeichen *von links nach rechts* gerechnet, ganz *ohne Vorrangregeln und ohne Klammern*[2].

Aufruf:

- Die Rechnung wird auf der *Befehlszeile* angegeben, das Endergebnis soll ausgegeben werden.
- *Jede Zahl und jedes Rechenzeichen ist ein einzelnes Eingabe-Wort* (d.h. du musst *Zwischenräume* vor und nach jedem Rechenzeichen eingeben!).

Hinweise:

- Das Programm soll mit **`double`**-Zahlen rechnen. (Welche Funktion brauchst du, um ein Wort von der Eingabe-Zeile in eine **`double`**-Zahl zu verwandeln?)
- Verwende für die Unterscheidung der 5 Rechenzeichen **`switch`** und **`case`** ! Bei ungültigen Rechenzeichen sollst du eine *Fehlermeldung* ausgeben und abbrechen.
- Für **^** : Die Funktion heißt **`pow`** und kommt aus **`math.h`**.
- *Prüfe* zu Beginn des Programms auf die richtige Anzahl von Eingabe-Worten: 1, 3, 5, ... Worte (ohne Programmnamen) sind ok, 0, 2, 4, ... können keine gültige Rechnung sein. (Warum?)

Tipp:

- Verwandle als erstes das *vorderste* Eingabe-Wort in eine Zahl und speichere diese als vorläufiges Zwischenergebnis.
- Mach dann eine *Schleife in **Zweierschritten*** über die restlichen Eingabe-Worte:
  - Lies zuerst einmal das *hintere* der beiden Worte als *neue Zahl*.
  - Im ersten Zeichen des *vorderen* Wortes muss das *Rechenzeichen* stehen: Mit **`argv[i][j]`** bekommst du das **`j`**-te Zeichen (einen **`char`**) im **`i`**-ten Wort (auch **`j`** beginnt bei 0).
  - Mach eine Fallunterscheidung je nach Rechenzeichen und rechne die neue Zahl dementsprechend zum bisherigen Zwischenergebnis dazu.

1 ***Achtung:*** Wenn du unter Linux arbeitest, musst du auf der Befehlszeile ***** statt ***** eintippen! Auch unter DOS/Windows müssen ***** und **^** speziell eingegeben werden. Am einfachsten ist es, in diesem Programm **`x`** statt ***** und **`p`** statt **^** zu verwenden!

2 Die Berücksichtigung von Klammern ist zu schwierig, die Beachtung der Vorrangregeln ist noch schwerer (wird erst im 3. Semester des Studiums gemacht...). Für beides braucht man sinnvollerweise Funktionen und Rekursion (==› später!).

3.) Suche nach rechtwinkeligen Dreiecken

Du stehst vor dem Problem, im Gelände *rechte Winkel* abzustecken. Dazu hast du ein Seil, das in regelmäßigen Abständen Knoten bzw. Markierungen hat. Du möchtest dieses Seil so im Dreieck zwischen drei Stöcken spannen, dass sich bei einem der Stöcke ein rechter Winkel ergibt. Jeder Stock soll dabei genau bei einem Knoten sein, d.h. auf jeder Seite des Dreieckes hat das Seil eine *ganze Anzahl von Abständen zwischen Knoten*.

Schreib ein Programm, das mit der Anzahl der Knoten im Seil aufgerufen wird und *alle möglichen (ganzzahligen!) Seitenlängen* ausgibt, bei denen ein rechter Winkel entsteht.

Beachte dabei, dass man das Seil nicht in voller Länge verwenden muss, sondern *auch kürzer* nehmen kann: Jeder Umfang *kleinergleich* der Seillänge ist als Lösung erlaubt. Denke auch scharf nach, was der maximale Umfang ist, d.h. wie viele Abstände ein Seil mit *n* Knoten hat.

- Dein Programm soll für *jede Lösung* eine Zeile mit dem *Umfang und den drei Seitenlängen* ausgeben, sowie ganz am Ende die *Gesamtzahl* aller Lösungen.
- Die Lösungen sollen in der *Reihenfolge aufsteigenden Umfangs* angezeigt werden.
- Um *keine doppelten Lösungen* anzuzeigen, sollen nur jene Lösungen berechnet werden, bei denen die *erste Seite die kürzeste und die letzte Seite die längste* ist (ganzzahlige gleichschenkelige rechtwinkelige Dreiecke gibt es ohnehin nicht).

Das Beispiel ist von den Programm-Konstrukten her ganz einfach (nur "ganz normale" Schleifen und **`if`**'s), aber betreffend Programmlogik (Lösungsidee) nicht sofort offensichtlich. Denk nach, wie du das Problem angehst, *bevor* du zu tippen beginnst!

Denkhilfen:

- Wenn die Ausgabe *nach steigendem Umfang geordnet* sein soll,
  was wird deine *äußerste Schleife* tun?
- Für jeden einzelnen Umfang musst du *alle Möglichkeiten für die erste Seite* durchprobieren, und für jeden einzelnen Wert von Umfang und erster Seite musst du *alle möglichen Werte für die zweite Seite* durchgehen.

  Nur für die dritte Seite musst du nichts mehr probieren, sie lässt sich *ausrechnen*, wenn der Umfang und die ersten beiden Seiten schon einen Wert haben.

  Wie viele ineinander geschachtelte Schleifen wirst du daher insgesamt haben?
- Überlege, wie die *Schleifenbedingungen bzw. Anfangs- und Endwerte* aussehen müssen, wenn nur Seitenlängen probiert werden sollen, bei denen die erste Seite kürzer als die zweite Seite und diese wiederum kürzer als die dritte Seite ist! Du sollst falsch geordnete Kombinationen nicht nachträglich mittels **`if`** überspringen, sondern *gleich gar nicht erzeugen* (oder gleich bei der ersten ein **`break`** machen)!
- Und ein bisschen Mathematik: Wie prüfst du,
  ob die gerade zu probierenden Seitenlängen ein rechtwinkeliges Dreieck bilden?

Hinweis:

Dieses Beispiel ist von der Logik her nicht mehr ganz einfach.
Bei den Schülern muss sich keiner Sorgen machen, wenn er das noch nicht schafft!