Programmieren 1 Übung: Einfache Funktionen...

Klaus Kusche

1.) Rekursive Ausgabe einer Zahl

Das Umwandeln eines **`int`** in eine Folge von Ziffern geschieht am einfachsten
von hinten nach vorne: Man dividiert die Zahl durch 10; der Divisionsrest ist die nächste Stelle von hinten nach vorne, und mit dem Divisionsergebnis rechnet man weiter.
Das wiederholt man, bis die Zahl 0 ist.

Leider kann man nicht von hinten nach vorne ausgeben, und auch das direkte Speichern von hinten nach vorne in einem String ist nur möglich, wenn man vorab weiss, wie viele Zeichen die Zahl im String schließlich belegen soll. Wenn man hingegen nur so viele Zeichen belegen will, wie die Zahl wirklich lang ist, muss man die Ziffern entweder aufwändiger von vorne nach hinten ausrechnen, oder man muss das Ergebnis zwischenspeichern und dann umdrehen oder verschieben.

Hier hilft die *Rekursion*, und zwar in Form folgender Idee für eine Funktion, die einen **`int`** genau so breit wie notwendig ausgibt[1]:

- Wenn die Zahl größer 9 ist, mach zuerst einen *rekursiven Aufruf*, der die *vorderen Stellen ausgibt* (d.h. die Zahl *ohne* die letzte Stelle)
- Dann gib die letzte Stelle aus.

Schreib eine *Funktion*, die genau das macht, und ein *Hauptprogramm* dazu, das alle Worte auf der Befehlszeile mittels **`atoi`** in **`int`**'s verwandelt und mit dieser Funktion einzeln zeilenweise ausgibt.

Zusatzaufgabe:

- Erweitere deine Funktion so, dass sie auch *negative* Zahlen richtig ausgibt.

2.) Der Binomialkoeffizient

Der Binomialkoeffizient[2] **"n über k"** zweier *nichtnegativer* ganzer Zahlen hat *3 Definitionen*, die alle zum selben Ergebnis führen:

- Eine **rekursive**: "n über k" = "(n – 1) über (k – 1)" + "(n – 1) über k"
- Eine unter Verwendung der **Fakultätsfunktion n!** : "n über k" = n! / (k! * (n – k)!)
- Eine als **Quotient zweier Produkte**: "n über k" =
  "Produkt von k Zahlen von n abwärts" / "Produkt von k Zahlen von 1 aufwärts"

("n über k" ist ganzzahlig, die Divisionen gehen sich *immer ohne Rest* aus)

Weiters gelten folgende *Randbedingungen*:

- "n über 0" = "n über n" = **1** (damit gilt auch "0 über 0" = 1)
- "n über k" = **0** für k › n oder k ‹ 0

1 Diese Idee ist zwar die eleganteste, aber nicht unbedingt die effizienteste!

2 Der Binomialkoeffizient hat viele Anwendungen in der Mathematik und Wahrscheinlichkeitsrechnung.

Schreib eine *Funktion*, die "n über k" berechnet, und dazu ein *Hauptprogramm*,
das zwei Zahlen n und k von der Befehlszeile einliest und das Ergebnis ausgibt.
Implementiere dabei nacheinander *alle drei Varianten*.

Für *negative* Zahlen oder wenn *n größer 12 ist* soll deine Funktion als *Ergebnis -1* liefern
(ab 13 kommt es bei der Variante mit Fakultät zu Überläufen, daher verbieten wir das!).

Bei der Variante mit der Fakultät:

- Was machst du in deinem Programm aus der *Fakultätsberechnung* sinnvollerweise?
- Die Fakultät rekursiv zu berechnen, ist zwar elegant,
  aber speicher- und performancemäßig nicht sinnvoll (Warum?).
  Bitte nimm dafür eine *Schleife*!

Zusatzaufgaben:

- Kannst du eine *globale Variable* definieren, in der du für die *rekursive Variante*
  die *Anzahl der Aufrufe mitzählst*, und diese dann nach der Berechnung
  im Hauptprogramm ausgeben?
- Um Rechenzeit bei der Variante mit *direkter Multiplikation* zu sparen:
  Nutze die Symmetrie des Binomialkoeffizienten ("n über k" = "n über (n – k)"),
  damit du im Zähler und im Nenner *nie mehr als je n/2 Multiplikationen* brauchst!
- Nutze deine Binomialfunktion, um ein *zweites Hauptprogramm* zu schreiben,
  das ein schönes *Pascal'sches Dreieck* (siehe z.B. Wikipedia) ausgibt
  (die gewünschte Höhe (Zeilenanzahl) wird auf der Befehlszeile angegeben).