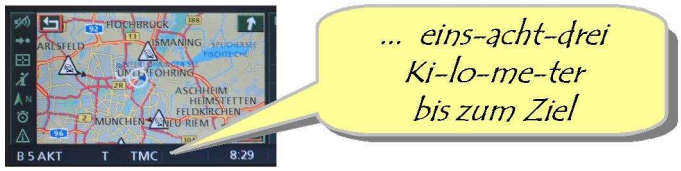
4. Algorithmus der Woche
Zahlen auf Deutsch aussprechen
Wie macht das ein Auto-Navigationssystem?

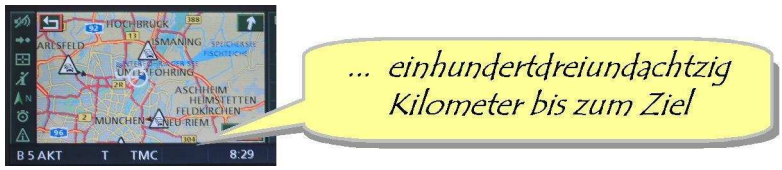
Autor
Lothar Schmitz, Universität der Bundeswehr München

Diese Woche geht es um das Aussprechen von Zahlen, wie das zum Beispiel ein Auto-Navigationssystem für jede benötigte Entfernungsangabe fertigbringt.

Die freundliche Stimme unseres Auto-Navis sagt dabei nicht wie ein alter Blech-Roboter:



Nein, sie spricht die Zahl genauso aus wie wir:



Ein „weltraumtaugliches“ Navigationssystem müsste noch viel größere Zahlen aussprechen können, z. B. die Zahl 12 345 678 987 654 321 als

zwölf Billiarden
dreihundertfünfundvierzig Billionen
sechshundertachtundsiebzig Milliarden
neunhundertsiebenundachtzig Millionen
sechshundertvierundfünfzigtausend
dreihunderteinundzwanzig

Sowas kann eigentlich jeder von uns. Aber wie sieht ein entsprechendes Programm aus? Da steckt der Teufel wie so oft im Detail! Wir präzisieren zuerst die Aufgabenstellung.

Aufgabe: Gegeben eine Zahl x mit $1 \leq x \leq 1024$. Erzeuge den Text, wie die Zahl auf Deutsch ausgesprochen wird! Wir gehen davon aus, dass Zahlen dieser Größe in der verwendeten Programmiersprache dargestellt werden können und dass Grundrechenarten wie Addition, Subtraktion, Multiplikation und Division (ganzzahlig mit Rest) zur Verfügung stehen.

Lösungsidee: Wie im Beispiel oben teilt man eine Zahl von rechts beginnend in Gruppen zu je 3 Ziffern auf:

Einer
Tausender
Millionen
Milliarden
Billionen
Billiarden
u. s. w.

Eine Zahl lässt sich wie folgt von rechts beginnend in Dreierzifferngruppen zerlegen:

```
1 i := −1    { i ist der Index der zuletzt gefundenen Gruppe }
2 while zahl ≥ 0 do
3     i := i + 1    { Index der neuen Gruppe }
4     gruppe[i] := zahl mod 1000    { Rest zur ... }
5     zahl := zahl / 1000    { Division durch 1000 }
6 endwhile
```

Anschließend enthält das Array gruppe die Zifferngruppen: die Einer in gruppe[0], die Tausender in gruppe[1], die Millionen in gruppe[2], die Milliarden in gruppe[3] ... Und in der Variablen *i* steht der Index der zuletzt gefundenen Gruppe.

Welche Namen es für große Zahlen gibt, kann man auf der Wikipedia-Seite Zahlennamen nachlesen. Dort erfährt man z. B., dass auf die Billiarden die Trillionen und Trilliarden folgen und dass eine Oktillion der Zahl 1048 enstpricht: das ist eine 1 mit 48 Nullen!

Der Rest des Verfahrens besteht nun darin, die Texte zu den Dreiergruppen zu erzeugen, und zwar beginnend mit der, die sich am weitesten links befindet. Wenn wir wissen, wie man den Zahltext zu einer Dreiergruppe erzeugt, dann ist die Aufgabe im Prinzip gelöst. Wenden wir uns also dieser Teilaufgabe zu!

Der Textanteil einer Dreiergruppe ist leer, wenn die Dreiergruppe den Wert 0 hat. Das zeigt ein Beispiel: 1 000 111 wird gesprochen als „eine Million einhundertelf“. Ansonsten zerlegt man die Dreiergruppe in Ihre Bestandteile: die Anzahl der Hunderter als Ziffer *h*, die Anzahl der Zehner als Ziffer *z* und die Anzahl der Einer als Ziffer *e*.

Spätestens an dieser Stelle muss man den Text zu jeder Ziffer kennen, um ihn ausgeben zu können. Da die Zahlen kleiner als 20 ziemlich unregelmäßig gebildet werden – z.B. „siebzehn“ statt „siebenzehn“ –, merken wir uns für jede dieser Zahlen den Text in einem Array:

```
kleiner20 : array [1..19] of String :=
[ „ein“, „zwei“, „drei“, „vier“, ... , „achtzehn“, „neunzehn“ ]
```

Für eine Zahl *i* zwischen 1 und 19 findet man den Text also in kleiner20[*i*]. Entsprechend verfahren wir mit den Zehnern (wobei für *i* zwischen 2 und 9 zehner[*i*] der Text zu $i \cdot 10$ ist):

```
zehner : array [2..9] of String :=
[ „zwanzig“, „dreißig“, „vierzig“, ... , „achtzig“, „neunzig“ ]
```

Wir kommen nun zum Kern des Verfahrens, der Funktion SPRICHDREIER, die den Text zu einer höchstens dreistelligen Zahl erzeugt. Der Text wird in der String-Variablen erg aufgesammelt: erg ist anfangs leer; Text anfügen kann man mit „&“. Zuerst wird die Hunderterstelle angefügt, dann der Rest. Ist der Rest < 20, dann entnimmt man den Text direkt dem Array kleiner20. Andernfalls fügt man an: Einerstelle - „und“ - Zehnerstelle (in dieser Reihenfolge!). Beachte, dass praktisch alles fehlen kann! So wird z. B. das „und“ zwischen Einern und Zehnern nur dann gebraucht, wenn sowohl Einer als auch Zehner vorhanden sind.

```
 1 function SPRICHDREIER(zahl)
 2     h := zahl / 100    { die Hunderter-Ziffer }
 3     r := zahl mod 100    { der Rest kleiner als 100 }
 4     z := r / 10    { die Zehner-Ziffer }
 5     e := zahl mod 10    { die Einer-Ziffer }
 6     erg := ""    { Text anfangs leer (bleibt leer, falls zahl = 0) }
 7 begin
 8         { zuerst vorhandene Hunderter anfügen: }
 9     if h > 0 then erg := erg & kleiner20[h] & "hundert"
10         { dann den Rest kleiner 100 umwandeln: }
11     if r > 0 then    { falls es diesen Rest überhaupt gibt }
12         if r < 20 then    { kleine Zahlen einfach aus der Tabelle nehmen: }
13             erg := erg & kleiner20[r]
14         else    { wenn 10er oder 1er fehlen, dann kein „und“! }
15             if e > 0 then erg := erg & kleiner20[e]    { Einer }
16             if e > 0 and z > 0 then erg := erg & "und"    { „und“ }
17             if z > 0 then erg := erg & zehner[z]    { Zehner }
18         endif
19     endif
20     return erg
21 end
```

Wandeln wir nach diesem Verfahren dreistellige Zahlen um, dann ergibt sich für viele Zahlen das gewünschte Resultat:

Für 111 ergibt sich „einhundertelf“.
Für 627 ergibt sich „sechshundertsiebenundzwanzig“.
Für 308 ergibt sich „dreihundertacht“.

In einigen Fällen ist das Resultat noch unbefriedigend:

Für 1 ergibt sich „ein“.
Für 901 ergibt sich „neunhundertein“.

Tatsächlich gibt es immer dann Probleme, wenn die Dreiergruppe mit „01“ endet. Die naheliegende Reparatur, in dem Array kleiner20 den String „ein“ durch ëinsßu ersetzen, funktioniert leider nicht: Z. B. für 41 ergäbe sich dann nämlich „einsundvierzig“. Noch mehr Fälle sind bezüglich der Endziffer 1 bei großen Zahlen zu berücksichtigen. Wir sagen ja:

„eintausend“ für 1 000,
„eine Million“ für 1 000 000,
„neunhunderteins Millionen“ für 901 000 000.

Die Lösung besteht aus zwei Maßnahmen:

1. In Dreiergruppen unterscheidet man zwischen Singular (für die 1), Plural und dem Sonderfall, dass die Dreiergruppe auf „01“ endet (z.B. in 901).
2. Die Anhänge an die Dreiergruppe unterscheidet man je nach dem, ob Singular oder Plural vorliegt und nach der Art der Dreiergruppe (Einer, Tausender, Millionen):
   - „eins“ und „neunhunderteins“
   - „eintausend“ und „neunhunderteinstausend“
   - „eine Million“ und „neunhunderteins Millionen“

Für die Anhänge verwenden wir dazu drei Arrays von Strings mit den unterschiedlichen Anhängen:

singular : array [1..8] of String :=
[“s“, “tausend“, “e Million“, “e Milliarde“, . . . ,“e Trilliarde“]

plural : array [1..8] of String :=
[““, “tausend “, “ Millionen“, “ Milliarden“, . . . ,“ Trilliarden“]

nulleins : array [1..8] of String :=
[“s“, “stausend “, “s Millionen“, “s Milliarden“, . . . ,“s Trilliarden“]

Damit kann man das allererste Programmstück wie folgt fortsetzen (in i steht immer noch der Index der höchstwertigen Dreiergruppe):

```
 1  text := ““
 2  while i ≥ 0 do
 3      if gruppe[i]> 0 then    { kein Textbeitrag, falls gruppe[i] = 0 }
 4          text := text & SPRICHDREIER(gruppe[i])
 5          if gruppe[i] = 1 then
 6              text := text & singular[i] & “ “
 7          else if gruppe[i] von der Form x01 then
 8              text := text & nulleins[i] & “ “
 9          else
10              text := text & plural[i] & “ “
11      endif
12      i := i – 1
13  endwhile
```

Danach enthält text wunschgemäß den Zahltext, wie man ihn im Deutschen spricht! Wer mag, kann probieren, das Programm z.B. an französische oder spanische Zahldarstellungen anzupassen.

Autor:

- Priv.-Doz. Dr. rer. nat. habil. Lothar Schmitz
  `http://www.unibw.de/inf2/Personen/Wissen_Mitarbeiter/lothar`

Externe Links:

- zusätzliche Informationsseite des Autors (dort ist der komplette Programmcode zu finden)
  `http://www.unibw.de/inf2/Personen/Wissen_Mitarbeiter/lothar/ADW/`
- Zahlennamen bei Wikipedia
  `http://de.wikipedia.org/wiki/Zahlennamen`