Rekursive Funktionen

Rekursive Funktionen

- Funktionen dürfen sich auch selber aufrufen.
- Damit lassen sich rekursive Algorithmen einfach implementieren.

Beispiel 1:

```
1  double power ( double x, int n )
2  {
3    if ( n == 0 )
4      return 1.0;
5    else
6      return x * power( x, n - 1 );
7  }
```

- Eine Rekursion braucht eine Abbruchbedingung (Zeile 3–4), sonst terminiert die Funktion nicht (Endlosschleife).

Iterative und rekursive Algorithmen

Rekursion und Iteration sind Konzepte zur Modellierung von Wiederholungen

- Iterative Algorithmen
  - Der Begriff „Iteration" stammt aus dem Lateinischen und bedeutet „wiederholen".
  - Die wiederholte Ausführung von Anweisungen als Schleife.

- Rekursive Algorithmen
  - Ein Problem wird auf ein einfacheres Teilproblem zurückgeführt und wird auf diese Weise letztlich gelöst.
  - Ein Programm ruft sich selbst auf.

Rekursive Algorithmen können systematisch in iterative umwandelt werden und umgekehrt.

Beispiel 1: Berechnung der Fakultät N!

```
unsigned int CalcFakultaet_Iterative(unsigned int n) {
	unsigned int fak = 1;
	if (n >= 1) {
		for (int i = 1; i <= n; i++) {
			fak *= i;
		}
	}
	return fak;
}

unsigned int CalcFakultaet_Rekursive(unsigned int n) {
	unsigned int fak = 1;
	if (n > 1) {
		fak = n * CalcFakultaet_Rekursive(n - 1);
	}
	return fak;
}
```

Definition [Bearbeiten | Quelltext bearbeiten]

Für alle natürlichen Zahlen n ist

$$n! = 1 \cdot 2 \cdot 3 \cdots n = \prod_{k=1}^{n} k$$

als das Produkt der natürlichen Zahlen von 1 bis n definiert. Da das leere Produkt stets 1 ist, gilt

$$0! = 1$$

Die Fakultät lässt sich auch rekursiv definieren:

$$n! = \begin{cases} 1, & n = 0 \\ n \cdot (n-1)!, & n > 0 \end{cases}$$

Rekursion bietet sich dann an, wenn man ein Problem durch den rekursiven Aufruf der
Funktion schrittweise in ein etwas einfacheres Problem umwandeln kann.

Beispiel 2: Ausgabe einer Zahl

Die Einerstelle einer Zahl kann mit der Modulo-Operation sehr einfach bestimmt werden, jedoch müssen zuvor die übrigen Ziffern der Zahl ausgegeben werden. Also wenn man 4285 ausgeben will ergibt sich folgender Ablauf:

```
Gib 428 aus:
   Gib 42 aus:
      Gib Ziffer 4 aus
      Gib Ziffer 2 aus.
   Gib Ziffer 8 aus.
Gib Ziffer 5 aus.
```

Pseudocode

```
GibZahlAus( zahl) {

  if (zahl > 9)
    GibZahlAus(zahl/10)

  Gib die Einerziffer aus
}
```

Vergleich

Vorteil von Rekursion
- Oftmals lassen sich Aufgaben leichter rekursiv formulieren, als iterativ.

Nachteile von Rekursion:
- Jeder Funktionsaufruf kostet Zeit. Also ist Rekursion prinzipiell langsamer als Iteration
- Jeder Funktionsaufruf kostet Speicherplatz. Also belegt die Rekursion viel Speicher auf dem Stack.