Inhalt

7 Rekursionen

7.1 Eine unendliche Geschichte

>>
Es war einmal ein Mann, der hatte sieben Söhne. Die sieben Söhne sprachen: „Vater erzähle uns eine Geschichte!“ Da fing der Vater an:

Es war einmal ein Mann, der hatte sieben Söhne. Die sieben Söhne sprachen: „Vater ...

Und wenn sie nicht gestorben sind, so erzählen sie noch heute.
<<

```
void geschichte()
{
  if( mannLebtNoch())
  {
    erzaehlung(); geschichte();          /* Rekursion */
  }
  return;
}
```

Rekursionen sind Funktionen, die sich selbst aufrufen.

Man unterscheidet:

Direkte Rekursionen: Funktionen, die sich selbst aufrufen.
Indirekte Rekursionen: Funktionen, die sich wechselseitig aufrufen.

In C sind direkte und indirekte Rekursionen erlaubt. Eine Ausnahme bildet hierbei die Funktion ***int main();***. Sie darf grundsätzlich nicht aufgerufen werden.

Rekursionen sind Rechenzeit und Speicherplatz intensiv.
Sie sollten deshalb vermieden werden.

```
void geschichte()
{
  while( mannLebtNoch())
  {
    erzaehlung();                  /* ohne Rekursion */
  }
  return;
}
```

7.2 Fakultät

Die Fakultät natürlicher Zahlen wird **iterativ** definiert:

$$0! = 1$$
$$n! = 1 \cdot 2 \cdot \ldots \cdot n = \prod_{k=1}^{n} k$$

```
unsigned long itFak( int n)
{
  unsigned long fak = 1;
  for( ; n > 1; fak *= n--);
  return fak;
}
```

Die Fakultät natürlicher Zahlen lässt sich **rekursiv** definieren:

$$0! = 1$$
$$n! = n \cdot (n-1)!$$

```
unsigned long rekFak( int n)
{
  if ( n) return n * rekFak( n - 1);
  return 1;
}
```

Parameterkellerung

rekFak(3)
PK: 3 * → **rekFak(2)**
PK: 2 * → **rekFak(1)**
3 **PK: 1 * →** **rekFak(0)**
2 **PK: 0**
3 1
2
PK ... Parameterkeller 3
3 * 2 ← 2 * 1 ← 1 * 1 ← 1 ←

fak.c

```
/*
 * Fakultaet iterativ und rekursiv
 */

# include <stdio.h>
/*
 * Iterative Berechnung der Fakultaet n!
 * Eingabewert: n
 * Rueckgabewert: n!
 */
unsigned long itFak( int);

/*
 * Rekursive Berechnung der Fakultaet n!
 * Eingabewert: n
 * Rueckgabewert: n!
 */
```

```
unsigned long rekFak( int);

int main()
{
  int n;
  do
  {
    do
    {             /* 13! = 6'227'020'800 > ULONG_MAX */
      printf( "Fakultaetsberechnung\n");
      printf( "Eingabe n = (n <= 12, 0 fuer ENDE) ");
      scanf( "%d", &n);
    } while( n < 0 || n > 12);

    if( n == 0) break;

    printf( "%d! = %lu\n", n, itFak( n));
    printf( "%d! = %lu\n", n, rekFak( n));
  } while( 1);

  return 0;
}


/*
 * Fakultaet iterativ
 */
unsigned long itFak( int n)
{
  unsigned long fak = 1;
  for( ; n > 1; fak *= n--);
  return fak;
}


/*
 * Fakultaet rekursiv
 */
unsigned long rekFak( int n)
{
  if ( n) return n * rekFak( n - 1);
  return 1;
}
```

fak.out

12! = 479 001 600 <
ULONG_MAX = 4 294 967 295 (4 Byte) <
13! = 6 227 020 800

=> Rechner: `13! = 1 932 053 504`

7.3 Türme von Hanoi

Türme von Hanoi des Monsieur Claus,
Anagramm des Mathematikers Lucas (vor 100 Jahren)

Legende: Zu Benares in Indien gibt es einen Tempel, in dem seit vielen Jahrhunderten Priester bemüht sind, einen Turm so umzulegen, wie Brahma es ihnen selbst geboten hat. Der Turm besteht aus 64 der Größe nach geordneten und auf einer Stange aufgesteckten dünnen goldenen Plättchen. Die Aufgabe lautet, den wenige Zentimeter hohen Turm von der ersten Stange auf die letzte Stange der drei zur Verfügung stehenden Stangen zu schaffen. Dabei darf jeweils nur eine Scheibe transportiert werden und niemals eine größere Scheibe auf eine kleinere gelegt werden. Sobald die Mönche dieses Ritual hinter sich gebracht haben, wird der Tempel zu Staub zerfallen und die Welt mit einem Donnerschlag untergehen! Wann wird wohl mit der indischen Götterdämmerung zu rechnen sein?

Trotz der indischen Geschichte ist das Puzzle als „Türme von Hanoi“ bekannt geworden.

void hanoi(int, int, int)

```
/*
 * hanoi: rekursive Funktion
 * Parameter: k .. Anzahl der Scheiben
 *            a .. Startstange
 *            b .. Zielstange
 *   1.  (k-1) Scheiben von Stange a nach Stange c
 *   2.   k. Scheibe von Stange a nach Stange b
 *   3.  (k-1) Scheiben von Stange c nach Stange b
 */

void hanoi( int k, int a, int b)
{
  if( k > 0)
  {
    hanoi( k - 1, a, 6 - a - b);              /* 1. */
    printf("%3d:%d => %d", k, a, b);          /* 2. */
    hanoi( k - 1, 6 - a - b, b);              /* 3. */
  }
  return;
}
```

Für 3 bzw. 4 Scheiben ergebnen sich die folgenden Bewegungen:

```
Anzahl der Scheiben: 3
Bewegung der einzelnen Scheiben:
--------------------------------
  1:1 => 3
  2:1 => 2
  1:3 => 2
  3:1 => 3
  1:2 => 1
  2:2 => 3
  1:1 => 3
```

k = 3 => 7 Bewegungen sind notwenig.

```
Anzahl der Scheiben: 4
Bewegung der einzelnen Scheiben:
--------------------------------
  1:1 => 2
  2:1 => 3
  1:2 => 3
  3:1 => 2
  1:3 => 1
  2:3 => 2
  1:1 => 2
  4:1 => 3
  1:2 => 3
  2:2 => 1
  1:3 => 1
  3:2 => 3
  1:1 => 2
  2:1 => 3
  1:2 => 3
```

k = 4 => 15 Bewegungen sind mindestens erforderlich.

Vermutung
Für alle $n \in N$ **gilt: Bei** n **Scheiben sind** $f(n) = 2^n - 1$ **Bewegungen erforderlich.**

Beweis: Sei M die Menge aller der natürlichen Zahlen, für welche die Behauptung gelten.

IA ($1 \in M$): Bei einer Scheibe ist nur eine Bewegung notwendig.

$$f(1) = 2^1 - 1 = 1$$

IS ($n \in M \rightarrow n' \in M$):

$$f(n+1) = 2 * f(n) + 1, \text{ wegen 1., 2. und 3. s.o.}$$
$$= 2 * (2^n - 1) + 1, \text{ nach IV}$$
$$= 2^{n+1} - 2 + 1 = 2^{n+1} - 1$$

Nach dem Induktionsaxiom gilt $M = N$ und damit die Vermutung.

Man benötigt bei n Scheiben mindestens $2^n - 1$ ***Züge, um das Problem zu lösen. Das sind bei 64 Scheiben 18'446'744'073'709'551'615 Aktionen.***

Die Priester werden wohl noch Milliarden von Jahren zu tun haben.