Rekursion in C#

Die Rekursion ist ein grundlegendes Verfahren der Informatik. Man kann sie als Wiederholung durch Schachtelung auffassen.

Aus der Mathematik kennt man z. B. die rekursive Definition von Zahlenfolgen, wie bei $a_{n+1} = a_n + 4$ und $a_0 = 3$.

Im Alltag kommt Rekursion in gewisser Form auch vor: Ein Kunde fragt in einem serviceorientierten Geschäft einen Verkäufer. Dieser kann die Frage nicht beantworten, deshalb fragt er einen Kollegen. Dieser kann die Frage ebenfalls nicht beantworten, deshalb fragt er einen weiteren Kollegen, bis der schließlich die Frage beantworten kann oder nicht mehr weiß, wenn er fragen soll.

In der Informatik tritt Rekursion ebenfalls sehr häufig auf. Ein typisches Beispiel ist die Suche nach Dateien in einem Verzeichnisbaum oder das Prinzip „teile und herrsche“ („divide and conquer“), bei dem man ein Problem so lange vereinfacht, bis es direkt lösbar ist.

Aufgabe 1:

Nenne Beispiele für Rekursion im Alltag, in der Mathematik oder in der Informatik.

Aufbau von rekursiven Methoden

Einfache rekursive Methoden sehen in der Regel folgendermaßen aus:

`void rekursive_Methode(…)`
`if(Abbruchbedingungen){`
`  // Anweisung(en)`
`}`
	Rekursionsanfang

`else{`
`  // weitere Anweisung(en)`
`  rekursive_Methode(…)`
`  // weitere Anweisung(en)`
`}`
	Rekursionsfall

Man unterscheidet zwei Fälle von Rekursion:

- Bei der **direkten Rekursion** ruft sich eine Methode selbst auf.
- Bei der **indirekten Rekursion** rufen sich zwei oder mehrere Methoden gegenseitig auf. Methode a ruft Methode b auf, die wiederum Methode a aufruft usw.

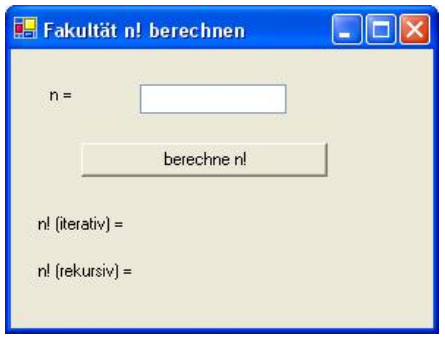
Bei der Rekursion muss man auf folgendes achten:

- Lokale Variablen werden bei jedem rekursiven Aufruf der Methode nochmals angelegt.
- Welcher Fall kann direkt gelöst werden, d. h. was ist mein Rekursionsanfang?
- Kann dieser Fall in **endlich vielen Schritten** erreicht werden?
- Wie komme ich vom Rekursionsanfang wieder zum ursprünglichen Problem?
- Aktionen, die nach dem Aufruf ausgeführt werden sollen, werden in der umgekehrten Reihefolge ausgeführt (vergleiche mit Aktenstapel).
- Die Methode darf sich nicht zu häufig selbst aufrufen, da es u. U. Speicherprobleme geben kann.
- Die Rekursion muss nach endlichen vielen Schritten beendet werden.
- Die Iteration, d.h. eine Schleife, ist häufig effektiver, aber Algorithmen bzw. Probleme lassen sich häufig einfacher rekursiv lösen.

Beispiel: Fakultät

Schauen wir uns als Beispiel die Fakultät von n an, die als $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$ definiert ist.


```
long fakultaet_it(long n){
  // Startwert für Schleife
  long ergebnis=1;

  for(long i=1;i<=n;i++){
    ergebnis=ergebnis*i;
  }

  return ergebnis;
}

long fakultaet_rek(long n){
  // Rekursionsanfang
  if(n==1){
    return 1;
  }
  else{  // Rekursionsfall
    long ergebnis=n*fakultaet_rek(n-1);
    return ergebnis;
  }
}

void Button1Click(object sender, System.EventArgs e)
{
   long n=long.Parse(textBoxN.Text);     // n einlesen

   long ergebnis_it=fakultaet_it(n);  // Methoden aufrufen
   long ergebnis_rek=fakultaet_rek(n);
   labelIterativ.Text=ergebnis_it.ToString();
   labelRekursiv.Text=ergebnis_rek.ToString();
}
```



Um größere Werte für die Fakultät berechnen zu können, wurde im Beispiel als Datentyp `long` statt `int` verwendet.
Zur Erinnerung an die iterative Lösung von frührer mit einer Schleife wurde die Methode `fakultaet_it` erstellt.
In der Methode `fakultaet_rek` wird die Fakultät rekursiv berechnet. Beim ersten Aufruf der Methode wird die Fakultät nicht direkt berechnet, sondern die Berechnung aufgeschoben. Die Grundidee ist dabei: „*Führe die Aufgabe in einfachere Version von sich selbst über*.“ In diesem Fall bedeutet dies, dass die Fakultät von n-1 berechnet wird, dann die Fakultät von n-2 usw. bis ein einfacher Fall erreicht wird, was bei 1! = 1 erreicht ist. Von diesem Punkt aus muss rückwärtsgelaufen werden, d.h. der Stapel der Methodenaufrufe muss rückwärts abgearbeitet werden. Damit wird dann 2!, dann 3!, dann 4! usw. berechnet bis n! erreicht ist.
Die Grundidee bei der rekursiven Berechnung der Fakultät erinnert an die vollständige Induktion aus der Mathematik.

Aufgaben:

2. Arbeite das oben abgedruckte Beispiel auf dem Papier für n = 6 durch.
3. Schreibe eine Methode, die die Summe $1 + 2 + 3 + \ldots + n$ iterativ berechnet und schreibe eine Methode, die zum Vergleich die gleiche Summe rekursiv berechnet.
4. Berechne die Quadratzahl n^2 einer eingegeben Zahl n nur durch Addition.
   Es gilt dabei: $1^2 = 1$ und $n^2 = (n-1)^2 + n + n - 1$
5. Schreibe eine rekursive Methode, die ein eingelesen Wort umdreht. Überlege dir vorher, was der Rekursionanfang und der Rekursionsfall ist.
6. Schreibe ein Programm, das die Fibonacci-Zahlen bis n (einlesen!) bestimmt.