

Der Call-Stack und die Rekursion

Ein populäres Beispiel für rekursive Algorithmen ist die Fakultätsfunktion:

```
5! = 5*4*3*2*1
fakultaet(5) = 120
fakultaet(3) = 3*2*1 = 6
```



(A1) Iterativ

Implementiere in BlueJ eine iterative Version der Fakultätsfunktion, die als Argument die Zahl entgegennimmt, deren Fakultät berechnet werden soll.



(A2) Rekursiv

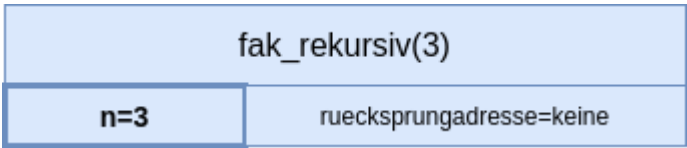
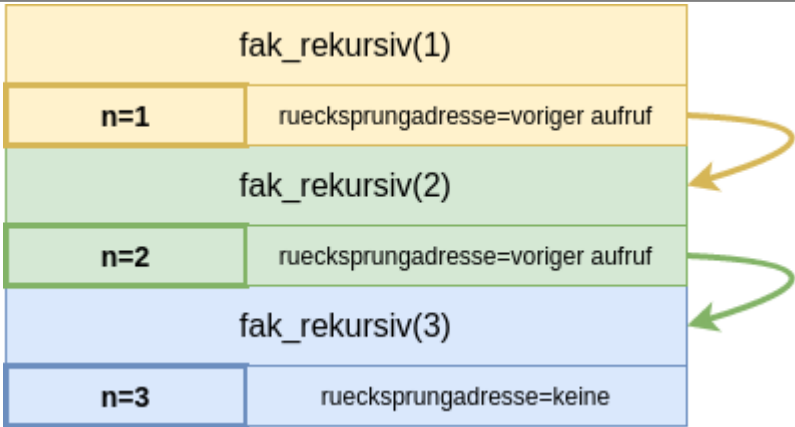
Implementiere anhand des folgenden Pseudocodes eine rekursive Version fak_rekursiv.

```
fak_rekursiv(int n):
  wenn n=1:
    return 1
  sonst:
    return n*fak_rekursiv(n-1)
```

- Was ist der Rekursionsfall, was der Basisfall?
- Teste deine rekursive Methode

Detaillierte Betrachtung des Call-Stacks bei der Rekursion

Was passiert	Wie sieht der Stack aus?

Was passiert	Wie sieht der Stack aus?
fak_rekursiv(3) wird aufgerufen. Auf dem Stack wird Speicher für diesen Aufruf reserviert. Es gibt keine Rücksprungadresse. Innerhalb dieses Aufrufs wird fak_rekursiv(2) (nächster Schritt) aufgerufen, da die Fallunterscheidung nicht zum Basisfall führt sondern zum Rekursionsfall.	 <pre> graph TD subgraph fak_rekursiv3 [fak_rekursiv(3)] n3[n=3] ra3[rücksprungadresse=keine] end </pre>
fak_rekursiv(2) wird aus dem vorhergehenden Aufruf heraus aufgerufen. Auf dem Stack wird Speicher für diesen Aufruf reserviert: Wichtig: Jeder Aufruf hat seinen eigenen Speicherbereich für Variablen, d.h. jeder Aufruf von fak_rekursiv hat sein eigenes n auf die die anderen Aufrufe nicht zugreifen können. Die Rücksprungadresse befindet sich jetzt im vorigen Aufruf der rekursiven Methode selbst. Das ist anders, als beim ersten Beispiel für den Call-Stack! Da erneut der Rekursionsfall eintritt, wird aus diesem Aufruf heraus fak_rekursiv(1) aufgerufen.	 <pre> graph TD subgraph fak_rekursiv2 [fak_rekursiv(2)] n2[n=2] ra2[rücksprungadresse=voriger aufruf] end subgraph fak_rekursiv3 [fak_rekursiv(3)] n3[n=3] ra3[rücksprungadresse=keine] end fak_rekursiv2 --> fak_rekursiv3 </pre>
fak_rekursiv(1) wird aus dem vorhergehenden Aufruf heraus aufgerufen. Auf dem Stack wird Speicher für diesen Aufruf reserviert. Die Rücksprungadresse befindet sich wiederum im vorigen Aufruf der rekursiven Methode selbst. Jetzt tritt der Basisfall ein!	 <pre> graph TD subgraph fak_rekursiv1 [fak_rekursiv(1)] n1[n=1] ra1[rücksprungadresse=voriger aufruf] end subgraph fak_rekursiv2 [fak_rekursiv(2)] n2[n=2] ra2[rücksprungadresse=voriger aufruf] end subgraph fak_rekursiv3 [fak_rekursiv(3)] n3[n=3] ra3[rücksprungadresse=keine] end fak_rekursiv1 --> fak_rekursiv2 fak_rekursiv2 --> fak_rekursiv3 </pre>

Was passiert	Wie sieht der Stack aus?
Die ist der erste Aufruf, der vollständig abgeschlossen ist. Es wird kein neuer Aufruf von fak_rekursiv auf den Call-Stack gelegt, sondern der Aufruf von fak_rekursiv(1) endet damit, dass 1 an die aufrufende Stelle zurückgegeben wird und die zum Aufruf gehörenden Daten vom Stack entfernt werden.	The diagram illustrates the state of the call stack. The top frame, <code>fak_rekursiv(1)</code>, is crossed out with a large red 'X', indicating it has been removed from the stack. Below it are two active frames: <code>fak_rekursiv(2)</code> (green) and <code>fak_rekursiv(3)</code> (blue). Each frame contains a sub-frame for the current value of <code>n</code> and the return address (<code>ruecksprungadresse</code>). <code>fak_rekursiv(1)</code> has <code>n=1</code> and <code>ruecksprungadresse=voriger aufruf</code>. <code>fak_rekursiv(2)</code> has <code>n=2</code> and <code>ruecksprungadresse=voriger aufruf</code>. <code>fak_rekursiv(3)</code> has <code>n=3</code> and <code>ruecksprungadresse=keine</code>. A yellow box labeled <code>return 1</code> has an arrow pointing to the <code>fak_rekursiv(1)</code> frame, indicating the return value. A green arrow points from the <code>fak_rekursiv(2)</code> frame to the <code>fak_rekursiv(3)</code> frame, indicating the next call.

From:
<https://info-bw.de/> -

Permanent link:
https://info-bw.de/faecher:informatik:oberstufe:algorithmen:rekursion:callstack_rekursion:start?rev=1642074493

Last update: 13.01.2022 11:48

