Imperative Programmierung

Übung 9: Rekursion

Justin Kreikemeyer[a]
Informatik, Uni Rostock

[a] Mit viel Inspiration von Andreas Ruscheinski

Leitfragen

- Was ist (Was ist (Was ist Rekursion?) Rekursion?) Rekursion?
- Wann und wofür verwendet man Rekursion?
- Wie geht das in der Praxis?

Rekursion: Überblick

- “Funktionen, die sich selber aufrufen” / “Verkleinere Problem, bis Lösung bekannt”
- Sehr wichtiges und zentrales Konzept zur Problemlösung in der Informatik
- Erst schwierig zu durchdringen, später aber extrem hilfreich

```c
#include <stdio.h>
void rec(int n) {
  printf("Was ist (");
  rec(n-1);                // darf er das? Ja, darf er!
  printf(" ) Rekursion?");
}
int main(int argc, char* argv[]) {
  rec(2);
  return 0;
}
```

Was gibt das oben stehende Programm aus?

Rekursion: Überblick

```c
#include <stdio.h>
void rec(int n) {
  if (n == 0) {                    // <- Abbruchkriterium!
    printf("Was ist Rekursion?");
    return;
  }
  printf("Was ist (");
  rec(n-1);                        // <- rekursiver Aufruf
  printf(" ) Rekursion?");
}
int main(int argc, char* argv[]) {
  rec(2);
  return 0;
}
```

So klappt es mit der Terminierung. Kleiner Exkurs: Was passiert eigentlich, wenn wir eine Funktion/Prozedur aufrufen?

Problemlösung mit Rekursion

- Zerlege Problem in kleinere(s) Problem(e)
- Mache das so oft, bis du eine triviale Lösung finden kannst
- Kombiniere die Teillösung(en) zur Gesamtlösung

Dieses Prinzip nennt man auch **Teile und Herrsche** (engl. divide and conquer)

Genauer Blick: Terminierung

Voraussetzungen:

1. Abbruchbedingung und *Rückgabe* von Triviallösung in diesem Fall
   - Parameter == bestimmte Konstante
   - Ende eines Arrays erreicht
   - ...
2. Rekursiver Aufruf auf "*kleinerem*" Problem; kleiner = näher an Triviallösung
   - Parameter - Konstante
   - Parameter / Konstante
   - Nächstes Array-Element
   - ...

Gemeinsames Beispiel

```c
#include <stdio.h>
void recFoo(int n) {
  if (n == 0) return 1;  // Triviallösung
  int a = recFoo(n - 1);     // rekursiver Aufruf ("Teile")
  int r = n*a;           // Kombination zu Gesamtlösung ("Herrsche")
  return r;
}
```

Beschreiben Sie den Ablauf des Programms!
Was gibt das oben stehende Programm aus?

Vorgehensweise bei der Implementierung

1. Definition der Funktion (Name, Parameter)
2. Überprüfung Abbruchbedinungen, Rückgabe Triviallösungen
3. Rekursiver Aufruf mit “kleinerer” Eingabe zum Teilen des Problems
4. Kombination der rekursiven Aufrufe zu kleinen Teilergebnissen
5. Rückgabe Ergebnis

Vorgehensweise bei der Implementierung

1. Definition der Funktion (Name, Parameter)

```
int isPrime(int numberToCheck, int divisor){
}
```

Vorgehensweise bei der Implementierung

2. Überprüfung Abbruchbedinungen, Rückgabe Triviallösungen

```
int isPrime(int numberToCheck, int divisor){
  if(numberToCheck <= 1) return 0;
  if(numberToCheck == 2) return 1;
  if(numberToCheck == divisor) return 1;
  if(numberToCheck % divisor == 0) return 0;
}
```

Vorgehensweise bei der Implementierung

3. Rekursiver Aufruf mit “kleinerer” Eingabe zum Teilen des Problems

```
int isPrime(int numberToCheck, int divisor){
  if(numberToCheck <= 1) return 0;
  if(numberToCheck == 2) return 1;
  if(numberToCheck == divisor) return 1;
  if(numberToCheck % divisor == 0) return 0;

  int otherCheckResult = isPrime(numberToCheck,divisor+1);
}
```

Vorgehensweise bei der Implementierung

4. Kombination der rekursiven Aufrufe zu kleinen Teilergebnissen
5. Rückgabe Ergebnis

```
int isPrime(int numberToCheck, int divisor){
  if(numberToCheck <= 1) return 0;
  if(numberToCheck == 2) return 1;
  if(numberToCheck == divisor) return 1;
  if(numberToCheck % divisor == 0) return 0;

  int otherCheckResult = isPrime(numberToCheck,divisor+1);

  int result = otherCheckResult;
  return result;
}
```

Fragen?

Aufgaben: Funktionen in C

Schreiben Sie die folgenden C-Programme und Funktionen[1]! Nutzen Sie dazu die Konzepte aus dieser Übung!
Bearbeitungszeit: bis 10 Minuten vor Schluss. Dann Besprechung von häufigen Problemen.

[1] Weitere Aufgaben können jederzeit beim Übungsleiter erfragt werden.

Rekursion I

RecRange Schreiben Sie eine Funktion `printNumbers(int n)`, die *mittels Rekursion* Zahlen von n bis 1 ausgibt.

RecRangeRev Schreiben Sie eine Funktion `printNumbersRev(int n)`, die *mittels Rekursion* die Zahlen von 1 bis n ausgibt.

RecPow Schrieben Sie eine Funktion `my_pow(int a, int n)`, welche mittels Rekursion a^n berechnet. Hinweis: $a^0 = 1$.

RecSeries Schreiben Sie eine Funktion `geoSeriesSum(int q, int n)`, welche *rekursiv* die geometrische Reihensumme $s_n = \sum_{k=0}^{n} q^k$ berechnet.

Rekursion II

Implementieren Sie einen äquivalenten Algorithmus mit Rekursion!

```
int countDigits(int num){
  int count = 0;
  while (num != 0) {
    num /= 10;
    count++;
  }
  return count;
}
```

Rekursion I

RecPalindrome Schreiben Sie eine Funktion `isPalindrome`, die *mittels Rekursion* prüft, ob eine gegebene Zeichenkette `s` ein Palindrom ist. Hinweis: welche "Hilfsargumente" werden ggf. benötigt?

RecFibonacci Die Fibonacci Reihe ist wie folgt definiert $fib(n+1) = fib(n) + fib(n-1)$ mit $fib(0) = 1$ und $fib(1) = 2$. Die nächste Zahl der Folge ergibt sich also durch die Summe der beiden vorherigen Zahlen. Implementieren Sie die Funktion `fib(int n)` mittels Rekursion.

RecCollatz Schreiben Sie mithilfe von Rekursion ein Programm, welche die Collatz-Folge für eine gegebene positive ganze Zahl generiert. Die Collatz-Folge ist wie folgt definiert:

1. Start: eine positive ganze Zahl n
2. Wenn das vorige Element gerade ist, ist das nächste Element die Hälfte des vorigen Elements; Wenn das vorige Element ungerade ist, ist das nächste Element 3 mal voriges Element plus 1.
3. Wenn das aktuelle Element 1 ist, endet die Folge.