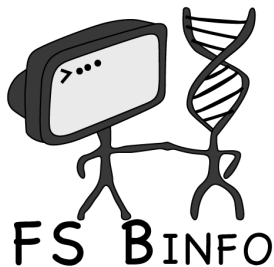


Rekursion vs. Iteration



Jeany Prinz

Fachschaft Bioinformatik

6. Dezember 2011

Zeitplan

- 25.10 Einführung in Linux und die Shell
- 08.11 Arbeiten mit Dateien auf der Shell
- 15.11. Shell 3 und Editor vim
- 22.11 Java-Einfuehrung mit Eclipse
- 29.11 Arrays und Schleifen
- 06.12 **Rekursion und Iteration**
- 13.12 File-I/O
- 20.12 Weihnachtsübung
- 10.01 Debuggen mit Eclipse
- 17.01 Objektorientierte Programmierung
- 24.01 Vererbung und Tipps
- 31.01 Quizz, Fragen, Evaluation

Webseite: [http://www.bioinformatik-muenchen.com/
bioinfocom/informatik-tutorium](http://www.bioinformatik-muenchen.com/bioinfocom/informatik-tutorium)



Rekursion

- lat. recurrere zurücklaufen
- Funktion wird durch sich selbst definiert
- Beispiel 1: Summe von 1 bis n

$$\text{sum}(n) = \begin{cases} 0 & \text{falls } n = 0 \\ \text{sum}(n - 1) + n & \text{sonst} \end{cases}$$

Abbruchbedingung
Rekursionsschritt

- Beispiel 2: grösster gemeinsamer Teiler

$$\text{ggT}(a, b) = \begin{cases} a & \text{falls } b = 0 \\ \text{ggT}(b, a \% b) & \text{sonst} \end{cases}$$

Abbruchbedingung
Rekursionsschritt

Iteration

- lat. wiederholen
- mehrfaches Ausführen einer Aktion
- Wiederholung durch Schleifen (for, while..), kein rekursiver Aufruf
- ggT iterativ:

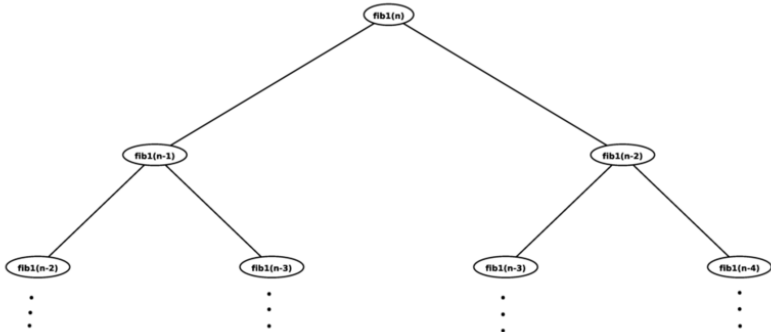
```
while (b>0){  
    int r=a % b;           Divisionsrest  
    a=b;                  neues a  
    b=r; }                neues b  
return a
```
- Übung: Implementiere sum iterativ

Fibonacci

- Folge von Zahlen (den Fibonacci-Zahlen), bei der sich die jeweils folgende Zahl durch Addition der beiden vorherigen Zahlen ergibt:
- 0, 1, 1, 2, 3, 5, 8, 13, ...
- Rekursive Definition

$$Fib(n) = \begin{cases} n & \text{falls } n = 0 \text{ oder } n = 1 \\ Fib(n-1) + Fib(n-2) & \text{sonst} \end{cases} \quad \begin{array}{l} \text{Abbruchbedingung} \\ \text{Rekursionsschritt} \end{array}$$

Aufrufbaum der rekursiven Fibonaccifunktion



Fibonacci iterativ

```
int prev1=0, prev2=1;
for(int i=0; i<n; i++) {
    int savePrev1 = prev1;
    prev1 = prev2;
    prev2 = savePrev1 + prev2;
}
return prev1;
```



Rekursion vs. Iteration

- bei Rekursion wird Programm immer wieder aufgerufen
⇒ Iteration ist meist effizienter
- Rekursive Lösungen legen u.a. die Werte der aktuellen Parameter und der lokalen Variablen auf dem Stack ⇒ benötigen mehr Speicher
- Rekursion ist intuitiver und meist übersichtlicher



Fakultät

- Fakultät(n) ist als das Produkt der natürlichen Zahlen von 1 bis n definiert
- $\prod_{i=1}^n i = 1 \cdot 2 \cdot \dots \cdot n$
- Aufgabe: Implementiere die Fakultät sowohl iterativ als auch rekursiv

Übung für zu Hause

- Schreibe ein rekursives und ein iteratives Programm das testet ob ein Wort ein Palindrom ist

Schönen Nikolaus

