

Exercise Week 08

GianAndrea Müller
mailto:muellegi@student.ethz

May 2, 2018

Exercise Week 08

GianAndrea Müller
mailto:muellegi@student.ethz

May 2, 2018

└ Time Schedule

Time Schedule

- 10' Rekursion - Konzept mit Beispiel
- 3' Rekursion vs Iteration
- 5' Rekursion - Optimierung
- 10' Rekursionsbäume
- 20' Beispielaufgaben zu Rekursion
- 15' Pause
- 45' Q&A

Time Schedule

- 10' Rekursion - Konzept mit Beispiel
- 3' Rekursion vs Iteration
- 5' Rekursion - Optimierung
- 10' Rekursionsbäume
- 20' Beispielaufgaben zu Rekursion
- 15' Pause
- 45' Q&A

Learning Objectives

- Verständnis: Rekursion

└ Rekursion - Konzept

- Was waere wenn sich eine Funktion selbst aufruft?
- Stack overflow.
- Wir muessen ein Ende definieren!

Rekursion - Konzept

```
1 int function(arg1, arg2){  
2     //Call function recursively  
3     function(arg1, arg2);  
4 }
```

Rekursion - Konzept

```
1 int function(arg1, arg2){  
2     //Call function recursively  
3     function(arg1, arg2);  
4 }
```

└ Rekursion - Konzept

- Loesung: Einfuehren einer Abschlussbedingung.
- Wichtig: Die Funktion muss so formuliert sein das die Abschlussbedingung zwingend erreicht wird.
- Es ist möglich das der stack overflow eintritt wird bevor die Abschlussbedingung eintritt.

Rekursion - Konzept

```
1 int function(arg1, arg2){  
2     // Termination condition  
3     if(terminate){  
4         return result;  
5     }  
6     // do stuff  
7  
8     //Call function recursively  
9     function(arg1, arg2);  
10 }
```

Rekursion - Konzept

```
1 int function(arg1, arg2){  
2     // Termination condition  
3     if(terminate){  
4         return result;  
5     }  
6     // do stuff  
7  
8     //Call function recursively  
9     function(arg1, arg2);  
10 }
```

└ Rekursion - Beispiel I

Rekursion - Beispiel I

```
1 double power(double x, int n){  
2   if (n == 1){  
3     return x;  
4   }  
5   return x*power(x, n-1);  
6 }  
7
```

Rekursion - Beispiel I

- Eine rekursive Implementation der Hochrechnung.
- Die Abbruchbedingung tritt ein sobald der letzte Faktor der Reihe erreicht wurde. (Man stelle sich die Hochrechnung als Reihe von Multiplikationen vor)
- Gegenuebergestellt mit der iterativen Implementation sieht das dann so aus:

```
1 double power(double x, int n){  
2   if (n == 1){  
3     return x;  
4   }  
5  
6   return x*power(x, n-1);  
7 }
```

└ Rekursion - Beispiel I

Rekursion - Beispiel I

```
1 double power(double x, int n){
2     if (n == 1){
3         return x;
4     }
5     return x*power(x, n-1);
6 }
7
8
9 double itpower(double x, int n){
10    double result = 1;
11    for(int i = 0; i<n; i++){
12        result *=x;
13    }
14    return result;
15 }
```

Rekursion - Beispiel I

```
1 double power(double x, int n){
2     if (n == 1){
3         return x;
4     }
5
6     return x*power(x, n-1);
7 }
8
9 double itpower(double x, int n){
10    double result = 1;
11    for(int i = 0; i<n; i++){
12        result *=x;
13    }
14    return result;
15 }
```

└ Rekursion vs Iteration

Vorteile und Nachteile der Rekursion

- Vorteile:
 - Komplexe Probleme können relativ einfach gelöst werden.
 - Der Code ist übersichtlicher.
- Nachteile:
 - Möglicher Stack Overflow
 - Langsamer
 - Debugging ist schwieriger.

Rekursion vs Iteration

Vorteile und Nachteile der Rekursion

- Vorteile:
 - ▶ Komplexe Probleme können relativ einfach gelöst werden.
 - ▶ Der Code ist übersichtlicher.
- Nachteile:
 - ▶ Möglicher Stack Overflow!
 - ▶ Langsamer
 - ▶ Debugging ist schwieriger.

└ Rekursion - Beispiel I - Optimierung

$$\begin{aligned}x^2 &= x \cdot x \\x^4 &= x^2 \cdot x^2 \\x^8 &= x^4 \cdot x^4 \\x^{16} &= x^8 \cdot x^8 \\x^{20} &= x^{16} \cdot x^4\end{aligned}$$

Rekursion - Beispiel I - Optimierung

- Die Potenz 20 kann mit nur 5 Multiplikationen gemacht werden. Das verlangt nach einer Optimierung unseres Algorithmus'.

$$x^2 = x \cdot x$$

$$x^4 = x^2 \cdot x^2$$

$$x^8 = x^4 \cdot x^4$$

$$x^{16} = x^8 \cdot x^8$$

$$x^{20} = x^{16} \cdot x^4$$

└ Rekursion - Beispiel I - Optimierung

- Das funktioniert aber nur für n die Potenzen von 2 sind.

Rekursion - Beispiel I - Optimierung

```
1 double power (double x, int n){
2     if (n == 1){
3         return x;
4     }
5
6     double temp = power(x, n/2);
7     return temp*temp;
8 }
```

Rekursion - Beispiel I - Optimierung

```
1 double power (double x, int n){
2     if (n == 1){
3         return x;
4     }
5
6     double temp = power(x, n/2);
7     return temp*temp;
8 }
```

└ Rekursion - Beispiel I - Optimierung

Rekursion - Beispiel I - Optimierung

```
1 double power (double x, int n){  
2   if (n == 1){  
3     return x;  
4   }  
5   else if (n%2 == 0) {  
6     double temp = power(x,n/2);  
7     return temp*temp;  
8   }  
9   else return x*power(x,n-1);  
10 }
```

Rekursion - Beispiel I - Optimierung

```
1 double power (double x, int n){  
2   if (n == 1){  
3     return x;  
4   }  
5   else if (n%2 == 0) {  
6     double temp = power(x,n/2);  
7     return temp*temp;  
8   }  
9   else return x*power(x,n-1);  
10 }
```

└ Rekursionsbäume

Rekursionsbäume

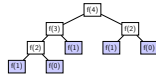
```
1 //POST: return value is the n-th
2 //Fibonacci number F(n)
3 unsigned int fib(const unsigned int n){
4     if (n == 0) return 0;
5     if (n == 1) return 1;
6     return fib(n-1) + fib(n-2); //n>1
7 }
```

Rekursionsbäume

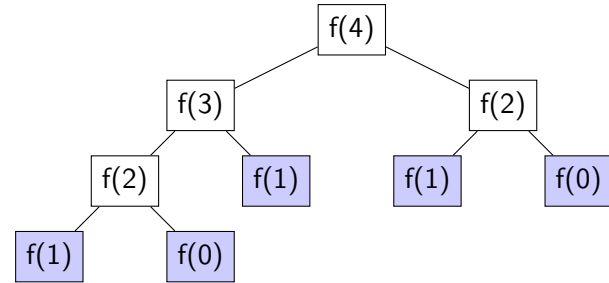
```
1 //POST: return value is the n-th
2 //Fibonacci number F(n)
3 unsigned int fib(const unsigned int n){
4     if (n == 0) return 0;
5     if (n == 1) return 1;
6     return fib(n-1) + fib(n-2); //n>1
7 }
```

Rekursionsbäume

└ Rekursionsbäume



- $\text{fib}(0)$ und $\text{fib}(1)$ werden insgesamt 5 mal aufgerufen!
- Die Anzahl blauer Kästchen entspricht der Fibonacci-Zahl, somit ist klar das die Anzahl Funktions- aufrufe strikt grösser als die resultierende Zahl ist.
- Für grössere n waechst die Fibonaccireihe exponentiell.
- Somit wird auch der Rechenaufwand extrem schnell höher!



└ Fibonacci Iterativ

- Die iterative Variante ist viel schneller.
- Trotzdem ist verzweigte Rekursion nicht generell zu vermeiden sondern kann sehr nützlich sein.

Fibonacci Iterativ

```

1 // POST: return value is the n-th
  Fibonacci number F(n)
2 unsigned int fib2 (const unsigned int n) {
3     if (n == 0) return 0;
4     if (n <= 2) return 1;
5     unsigned int a = 1; // F_1
6     unsigned int b = 1; // F_2
7     for (unsigned int i = 3; i <= n; ++i) {
8         unsigned int a_prev = a; // F_i-2
9         a = b; // F_i-1
10        b += a_prev; // F_i-1 += F_i-2 -> F_i
11    }
12    return b;
13 }

```

Fibonacci Iterativ

```

1 // POST: return value is the n-th
  Fibonacci number F(n)
2 unsigned int fib2 (const unsigned int n) {
3     if (n == 0) return 0;
4     if (n <= 2) return 1;
5     unsigned int a = 1; // F_1
6     unsigned int b = 1; // F_2
7     for (unsigned int i = 3; i <= n; ++i) {
8         unsigned int a_prev = a; // F_i-2
9         a = b; // F_i-1
10        b += a_prev; // F_i-1 += F_i-2 -> F_i
11    }
12    return b;
13 }

```

└ Rekursion - Aufgabe I ~ 5'

Rekursion - Aufgabe I ~ 5'

Aufgabe

• Schreiben Sie die folgende Funktion in iterativer Form.

```
1 unsigned int f (const unsigned int n)
2 {
3     if (n <= 2) return 1;
4     return f(n-1) + 2*f(n-3);
5 }
```

Rekursion - Aufgabe I ~ 5'

Aufgabe

- Schreiben Sie die folgende Funktion in iterativer Form.

```
1 unsigned int f (const unsigned int n)
2 {
3     if (n <= 2) return 1;
4     return f(n-1) + 2*f(n-3);
5 }
```

└ Rekursion - Lösung I

Rekursion - Lösung I

```

1 unsigned int f_it (const unsigned int n) {
2     if(n<=2) return 1;
3     unsigned int a = 1; // f(0)
4     unsigned int b = 1; // f(1)
5     unsigned int c = 1; // f(2)
6     for (int i = 3; i<n; ++i){
7         int a_prev = a    // f(i-3)
8         a = b;           // f(i-2)
9         b = c;           // f(i-1)
10        c = b + 2*a_prev; // f(i)
11    }
12    return c + 2*a;
13 }

```

Rekursion - Lösung I

```

1 unsigned int f_it (const unsigned int n) {
2     if(n<=2) return 1;
3     unsigned int a = 1; // f(0)
4     unsigned int b = 1; // f(1)
5     unsigned int c = 1; // f(2)
6     for (int i = 3; i<n; ++i){
7         int a_prev = a    // f(i-3)
8         a = b;           // f(i-2)
9         b = c;           // f(i-1)
10        c = b + 2*a_prev; // f(i)
11    }
12    return c + 2*a;
13 }

```


└ Rekursion - Aufgabe II ~ 5'

Rekursion - Aufgabe II ~ 5'

Aufgabe

• Schreiben Sie die folgende Funktion in iterativer Form.

```
1 unsigned int f (const unsigned int n)
2 {
3     if (n==0) return 1;
4     return f(n-1) + 2*f(n/2);
5 }
```

Rekursion - Aufgabe II ~ 5'

Aufgabe

- Schreiben Sie die folgende Funktion in iterativer Form.

```
1 unsigned int f (const unsigned int n)
2 {
3     if (n==0) return 1;
4     return f(n-1) + 2*f(n/2);
5 }
```

└ Rekursion - Lösung II

Rekursion - Lösung II

```
1 unsigned int f_it (const unsigned int n)
2 {
3     if (n==0) return 1;
4
5     std::vector<unsigned int> f_val(n+1,0);
6     f_val[0] = 1;
7     for (int i = 1; i<=n; ++i)
8         f_val[i] = f_val[i-1] + 2*f_val[i/2];
9
10    return f_val[n];
11 }
```

Rekursion - Lösung II

```
1 unsigned int f_it (const unsigned int n)
2 {
3     if (n==0) return 1;
4
5     std::vector<unsigned int> f_val(n+1,0);
6     f_val[0] = 1;
7     for (int i = 1; i<=n; ++i)
8         f_val[i] = f_val[i-1] + 2*f_val[i/2];
9
10    return f_val[n];
11 }
```

└ Beispiel: Self assessment

Beispiel: Self assessment

```

1 int f(const int* begin, const int* end){
2     int val1 = *begin;
3     if(++begin != end){
4         const int val2 = f(begin,end);
5         if(val2>val1) { val1 = val2;}
6     }
7     return val1;
8 }
9
10 int g(const int *begin, const int *end){
11     int val = *begin;
12     for (const int * it=++begin; it != end;
13         ++it){
14         if (*it > val){ val = *it}
15     }
16     return val
17 }

```

Beispiel: Self assessment

```

1  int f(const int* begin, const int* end){
2      int val1 = *begin;
3      if(++begin != end){
4          const int val2 = f(begin,end);
5          if(val2>val1) { val1 = val2;}
6      }
7      return val1}
8
9  int g(const int *begin, const int *end){
10     int val = *begin;
11     for (const int * it=++begin; it != end;
12         ++it){
13         if (*it > val){ val = *it}
14     }
15     return val
16 }

```