Übungsstunde — Informatik — 07

Adel Gavranović

Follow-up, ASCII-Zeichen, `char`, Rekursion, viel **code** expert

Übersicht

Follow-up
- `const` und Funktionen
- Repetition: Fliesskommazahlen

(ASCII) Zeichen in C++ (`char`)
Rekursion
Feedback
Alte Prüfungsfragen

`n.ethz.ch/~agavranovic`

Material

Webpage

Mail

1. Follow-up

Follow-up aus letzter Übungsstunde

- Nicht viel Follow-up heute
- Habe ich etwas vergessen?
- Werde mir in Zukunft Fragen besser notieren…

1. Follow-up

1.1. `const` und Funktionen

`const` und Funktionen

1.
```
void Funktion(const Type& n){                    //  const als Arugment
    // n kann nicht verändert werden
} // sehr oft sinnvoll; muss man kennen!
```

2.
```
const Type& Funktion(const Type& n){             //  const als Return
    // n kann nicht verändert werden
    return n; // ursprüngliches n wird returned
} // manchmal sinnvoll; ziemlich spezifish
```

3.
```
void Funktion(Type n) const {                    // const-Funktionen
    // n kann verändert werden
    // nichts in der Klasse, der die Funktion angehört,
    // kann verändert werden. [Klassen wurden noch nicht behandelt]
} // oft sinnvoll; wird vielleict besprochen im Thema "Klassen"
```

Fragen/Unklarheiten?

1. Follow-up

1.2. Repetition: Fliesskommazahlen

Fliesskommazahlensysteme

Aufgabe

- Versucht, folgende Aufgaben zu lösen
- Fragt, wenn etwas unklar ist

Informatik
Exercise Session

Consider the normalized floating point number system $F^*(\beta, p, e_{\min}, e_{\max})$ with $\beta = 2$, $p = 3$, $e_{\min} = -4$, $e_{\max} = 4$.
Compute the following expressions as the parentheses suggest, representing each intermediate result (and the final result) in the normalized floating point system according to the rules of computing with floating point numbers.

$(10 + 0.5) + 0.5$

	decimal		binary
	10		?????
+	0.5		?????
=			?????
+	0.5		?????
=	??	←	?????

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		?????
+	0.5		?????
=			?????
+	10		?????
=	??	←	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			?????
$+$	0.5		?????
$=$	??	$\leftarrow$	?????

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		?????
$+$	0.5		?????
$=$			?????
$+$	10		?????
$=$	??	$\leftarrow$	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.0101 \cdot 2^3$
$+$	0.5		?????
$=$	??	$\leftarrow$	?????

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		?????
$+$	0.5		?????
$=$			?????
$+$	10		?????
$=$	??	$\leftarrow$	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	??	$\leftarrow$	?????

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		?????
$+$	0.5		?????
$=$			?????
$+$	10		?????
$=$	??	$\leftarrow$	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	10	$\leftarrow$	$1.01 \cdot 2^3$

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		?????
$+$	0.5		?????
$=$			?????
$+$	10		?????
$=$	??	$\leftarrow$	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	10	$\leftarrow$	$1.01 \cdot 2^3$

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		$1.00 \cdot 2^{-1}$
$+$	0.5		$1.00 \cdot 2^{-1}$
$=$			?????
$+$	10		?????
$=$	??	$\leftarrow$	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	10	$\leftarrow$	$1.01 \cdot 2^3$

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		$1.00 \cdot 2^{-1}$
$+$	0.5		$1.00 \cdot 2^{-1}$
$=$			$1.00 \cdot 2^0$
$+$	10		$1010.00 \cdot 2^0$
$=$	??	$\leftarrow$	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	10	$\leftarrow$	$1.01 \cdot 2^3$

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		$1.00 \cdot 2^{-1}$
$+$	0.5		$1.00 \cdot 2^{-1}$
$=$			$1.00 \cdot 2^0$
$+$	10		$1010.00 \cdot 2^0$
$=$	??	$\leftarrow$	$1011.00 \cdot 2^0$

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	10	$\leftarrow$	$1.01 \cdot 2^3$

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		$1.00 \cdot 2^{-1}$
$+$	0.5		$1.00 \cdot 2^{-1}$
$=$			$1.00 \cdot 2^0$
$+$	10		$1010.00 \cdot 2^0$
$=$	??	$\leftarrow$	$1.011 \cdot 2^3$

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	10	$\leftarrow$	$1.01 \cdot 2^3$

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		$1.00 \cdot 2^{-1}$
$+$	0.5		$1.00 \cdot 2^{-1}$
$=$			$1.00 \cdot 2^0$
$+$	10		$1010.00 \cdot 2^0$
$=$	12	$\leftarrow$	$1.10 \cdot 2^3$

Fragen/Unklarheiten?

2. Feedback zu **code** expert

Allgemeines bezüglich Feedback/Fragen

...

Allgemeines bezüglich **code** expert

...

Fragen bezüglich **code** expert eurerseits?

3. Lernziele

Ziele für Heute

Lernziele

- ☐ Programme, die `char` verwenden, tracen und schreiben können
- ☐ Sich mit dem Konzept der Rekursion vertraut gemacht

4. Zusammenfassung

5. (ASCII) Zeichen in C++ (`char`)

Übung "Converting Input to UPPER CASE"

Aufgabe

1. Überlegt, wie ihr die Aufgabe "Converting Input to UPPER CASE" auf **code** expert am besten angeht
2. Programmiert (optional gruppenweise) eine Lösung

Übung "Converting Input to UPPER CASE"

Aufgabe

Schreibt ein Programm, das eine durch ein Zeilenumbruchszeichen begrenzte Zeichenfolge als Vektor von `char` einliest. Anschließend soll das Programm die Folge ausgeben, wobei alle Kleinbuchstaben in GROSSBUCHSTABEN umgewandelt werden.

"Converting Input to UPPER CASE" — Lösung

```
#include <iostream>
#include <vector>
#include <ios>        // not really needed, don't worry about it
```

"Converting Input to UPPER CASE" — Lösung

```
// POST: Converts the letter to upper case.
void char_to_upper(char& letter){
    int shift_distance = 'a' - 'A';     // 'a' > 'A' (if conv. to ints)
                                        // distance between the upper
                                        // and lower case numbers
    if('a' <= letter && letter <= 'z'){
        letter -= shift_distance;
    }
}

// POST: Converts all letters to upper-case.
void to_upper(std::vector<char>& letters){
    for(int i = 0; i < letters.size(); ++i){
        char_to_upper(letters.at(i));
    }
}
```

"Converting Input to UPPER CASE" — Lösung

```
std::vector<char> letters;
char ch;

// Step 1: Read input.
do {
    std::cin >> ch;
    letters.push_back(ch);
} while(ch != '\n');

// Step 2: Convert to upper-case.
to_upper(letters);

// Step 3: Output.
for(int i = 0; i < letters.size(); ++i){
    std::cout << letters.at(i);
}
```

Fragen/Unklarheiten?

6. Rekursion

Alte Prüfungsaufgabe

Eckdaten

- Prüfung: 01.2022 Computer Science (MATH/PHYS/RW)
- Einfache Rekursionsaufgabe
- Gesamtpunkte in der Prüfung: 85 Punkte
- Gesamtzeit für die Prüfung: 120 minutes
- Punkte für die Aufgabe: 5 Punkte
- Geschätzte Zeit für die Aufgabe: 7 minutes $= 120 * (5/85)$

Kleine Prüdungssimulation

- Schalten in den "Prüfungsmodus" und stellt alles was ihr brauchen könnten bereit
- Öffnet "[Exam 2022.01 (MATH/PHYS/RW)] Compute Series" auf **code** expert
- Programmiert eine (rekursive) Lösung
- Zeit: 7 Minuten

Alte Prüfungsaufgabe

Wir wollen eine Funktion mit den folgenden PRE und POSTs schreiben

```
// PRE:   a positive integer n
//
// POST:  returns the n-th number of a series x_n, defined as
//        x_n = 2,                      for n = 1
//        x_n = 1,                      for n = 2
//        x_n = x_(n-1) + x_(n-2),      for n > 2
//
// Example:
//  * n == 1 ~~> 2
//  * n == 2 ~~> 1
//  * n == 3 ~~> 3
```

Alte Prüfungsaufgabe — Lösung

```
// PRE:   a positive integer n
//
// POST:  returns the n-th number of a serie x_n, defined as
//        x_n = 2,                    for n = 1
//        x_n = 1,                    for n = 2
//        x_n = x_(n-1) + x_(n-2),    for n > 2

int compute_element(int n) {
    if (n == 1) {
        return 2;
    } else if (n == 2) {
        return 1;
    } else {
        return compute_element(n-1) + compute_element(n-2);
    }
}
```

Fragen/Unklarheiten?

Übung "Partial Sum"

Aufgabe

Schreibe eine Funktion, die

1. die Summe aller natürlichen Zahlen kleiner oder gleich n per Rekursion berechnet und diesen Wert zurückgibt
2. alle addierten Terme in aufsteigender Reihenfolge (von 0 bis n) in der gleichen Rekursiven Funktion ausdruckt

Übung "Partial Sum"

- Öffnet "Partial Sum" auf **code** expert
- Überlegt euch, wie ihr das Problem mit Stift und Papier angehen würdet
- Programmiert eine (rekursive) Lösung (optional in Gruppen)

"Partial Sum" — Lösung

```cpp
int partial_sum(const int n) {
  if (n == 0) {
    return 0;
  } else {
    // print descending
    // std::cout << n << std::endl;

    int partial = partial_sum(n - 1);

    // print ascending
    std::cout << n << std::endl;

    return n + partial;
  }
}
```

"Partial Sum" — Lösung

```
int main() {
  std::cout << "n = ";

  int n;
  std::cin >> n;

  std::cout << partial_sum(n) << std::endl;

  return 0;
}
```

Fragen/Unklarheiten?

Übung "Power Function"

Frage

Wie viele Rekursionsaufrufe braucht die folgende Funktion, um x^7 zu berechnen?

```
int power(const int x, const int n) {

    if (n == 0){
        return 1;
    } else if (n == 1) {
        return x;
    }

    return x * power(x, n - 1);
}
```

Antwort: 7

Übung "Power Function"

- Öffnet "Power Function" auf **code** expert
- Überlegt euch, wie ihr das Problem mit Stift und Papier angehen würdet
- Programmiert eine (rekursive) Lösung
- *Tipp: Die Aufgabe ist eine Verallgemeinerung der Aufgabe "Multiply with 29" der ersten Woche*

"Power Function" — Lösung

```
// POST: result == x^n
int power (const int x, const int n) {
  if(n == 0) {
    return 1;
  } else if(n == 1) {
    return x;
  } else if(n % 2 == 0) {        // case n = 2m for some m in N
    int temp = power(x, n/2);    // temp, to not call the function twice!
    return temp * temp;          // since x^n = x^(2m) = x^m * x^m
  } else {
    return x * power(x, n-1);
  }
}
```

Fragen/Unklarheiten?

Die Türme von Hanoi

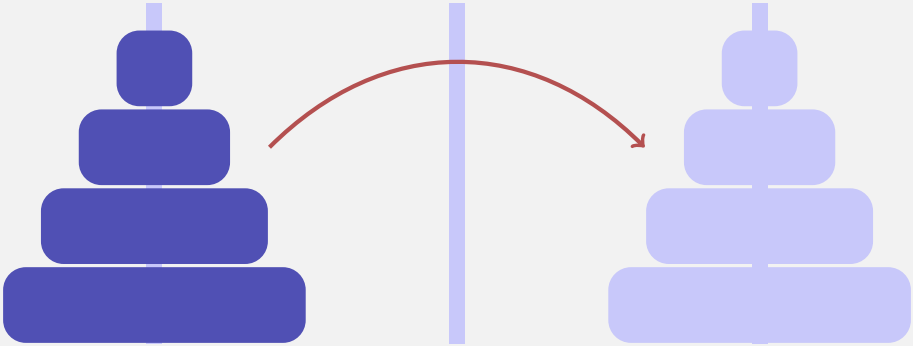
Mit dieser Übung zu kämpfen, ist ein lang gehegte Tradition unter Programmieranfänger:innen.

Sie ist berüchtigt schwierig, wenn man mit Rekursion nicht vertraut ist.

Experiment: Die Türme von Hanoi

Links Mitte Rechts

Experiment: Die Türme von Hanoi

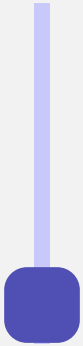


Die Türme von Hanoi - So gehts!

Links Mitte Rechts

Die Türme von Hanoi - So gehts!

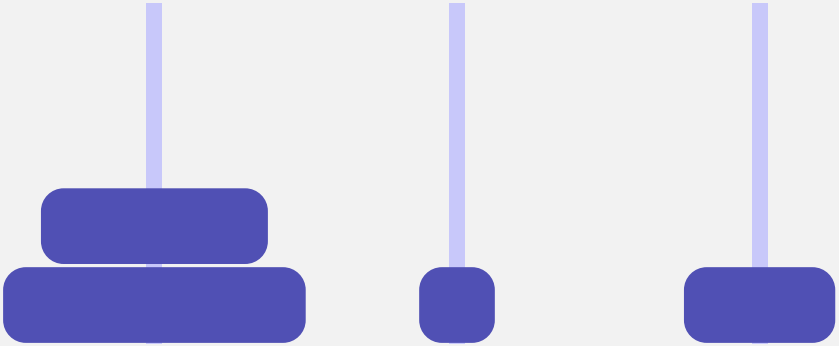
Links



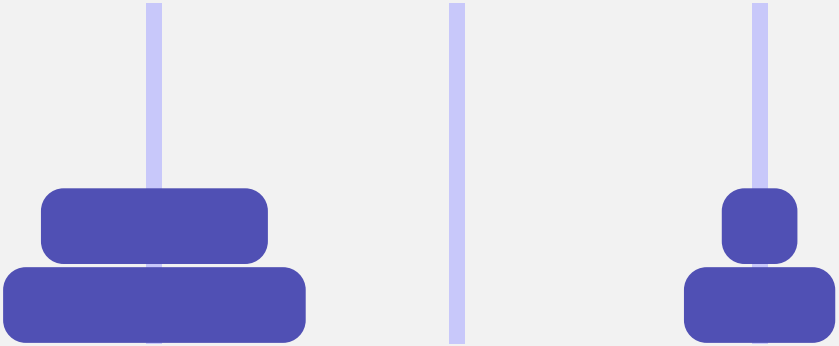
Mitte

Rechts

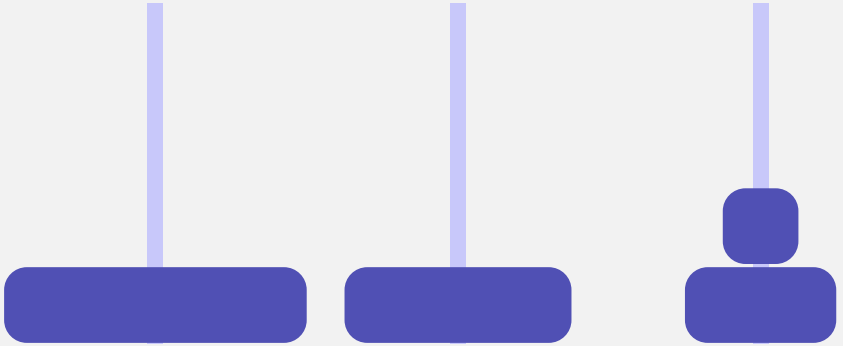
Die Türme von Hanoi - So gehts!



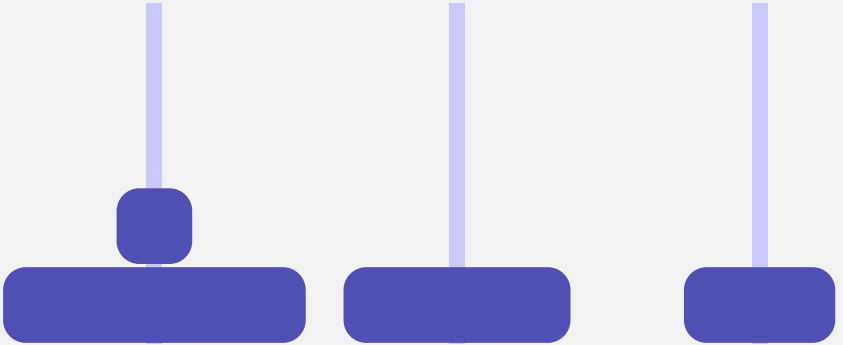
Die Türme von Hanoi - So gehts!



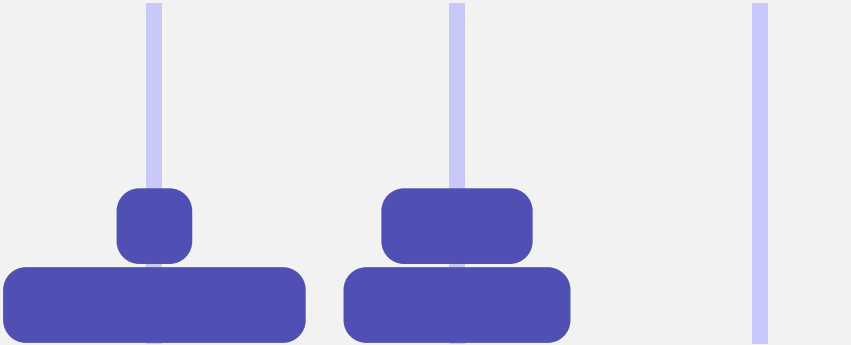
Die Türme von Hanoi - So gehts!



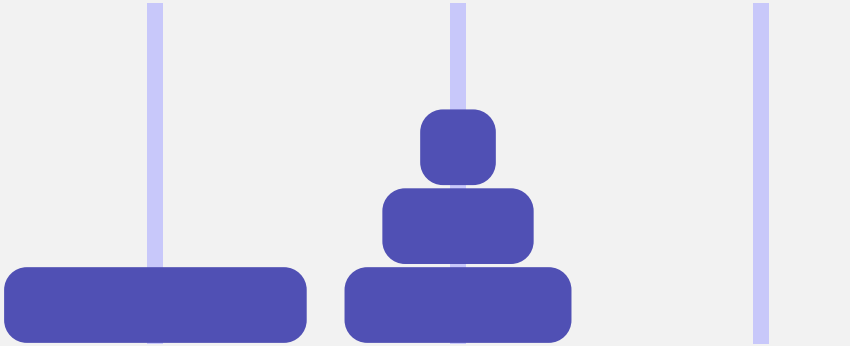
Die Türme von Hanoi - So gehts!



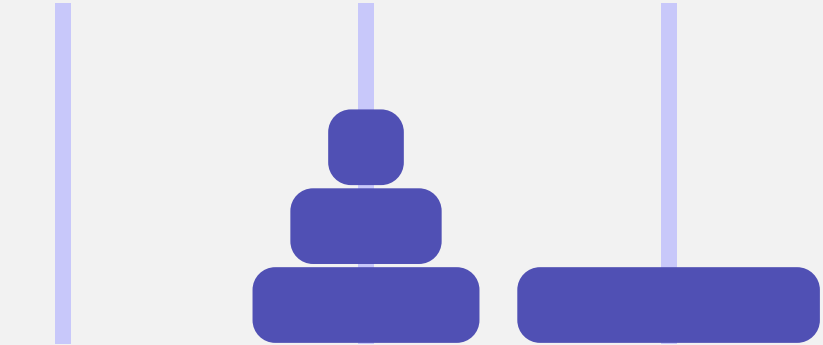
Die Türme von Hanoi - So gehts!



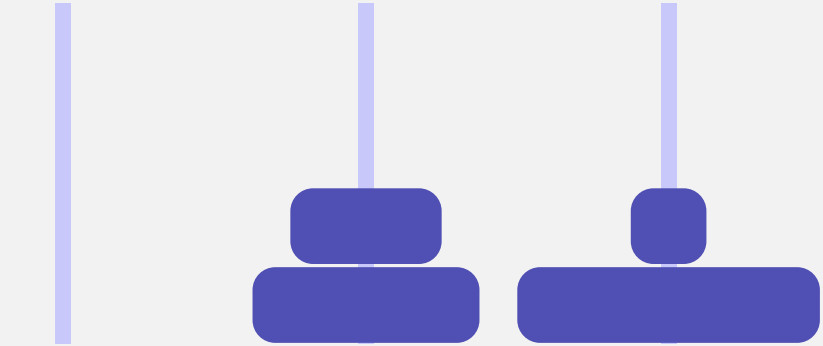
Die Türme von Hanoi - So gehts!



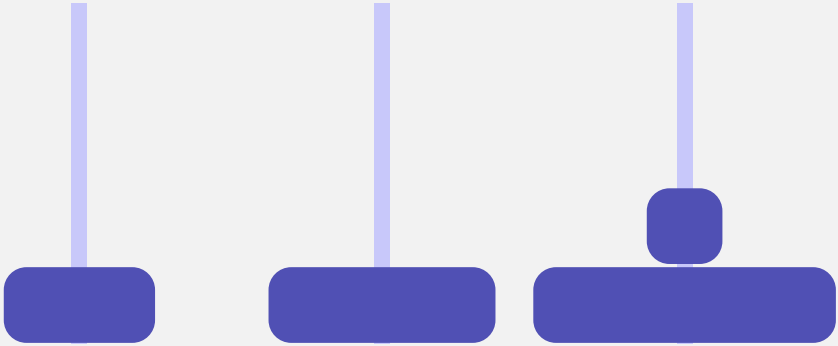
Die Türme von Hanoi - So gehts!



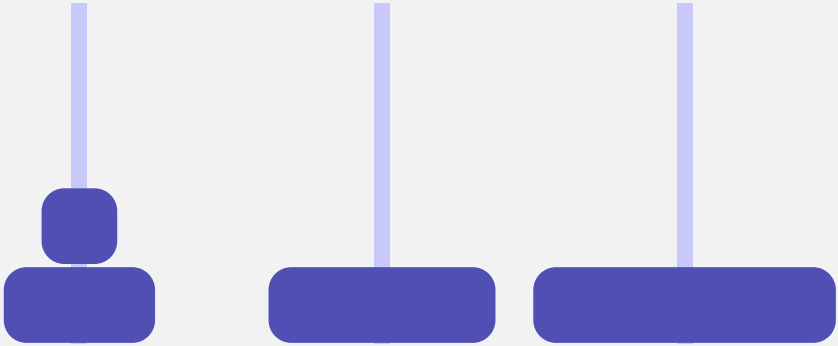
Die Türme von Hanoi - So gehts!



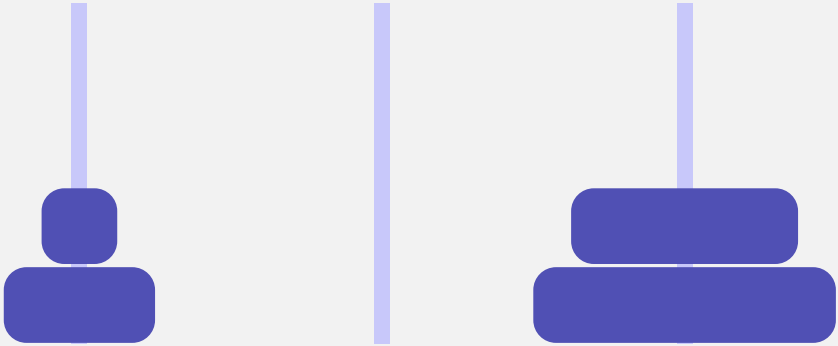
Die Türme von Hanoi - So gehts!



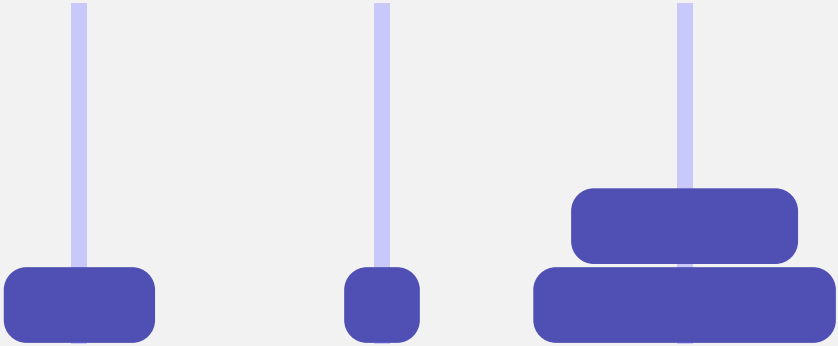
Die Türme von Hanoi - So gehts!



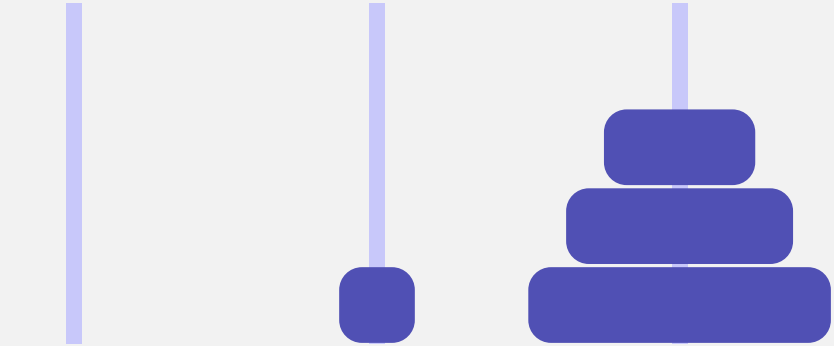
Die Türme von Hanoi - So gehts!



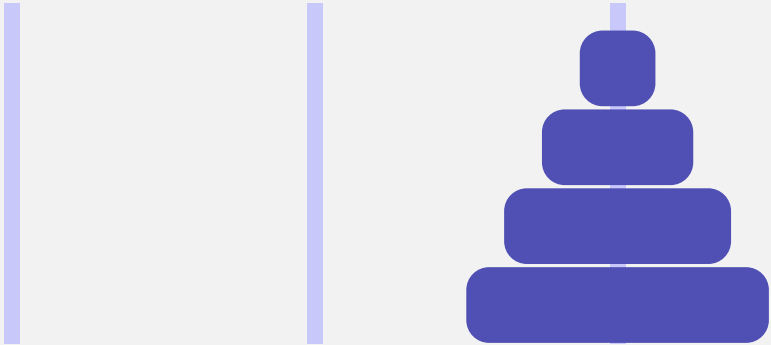
Die Türme von Hanoi - So gehts!



Die Türme von Hanoi - So gehts!



Die Türme von Hanoi - So gehts!

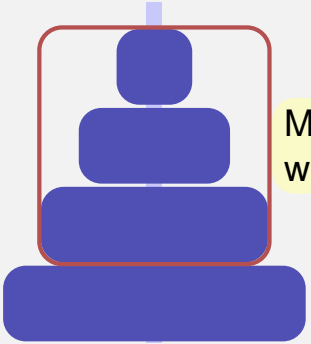


Links Mitte Rechts

Die Türme von Hanoi - Rekursiver Lösungsansatz

Links Mitte Rechts

Die Türme von Hanoi - Rekursiver Lösungsansatz

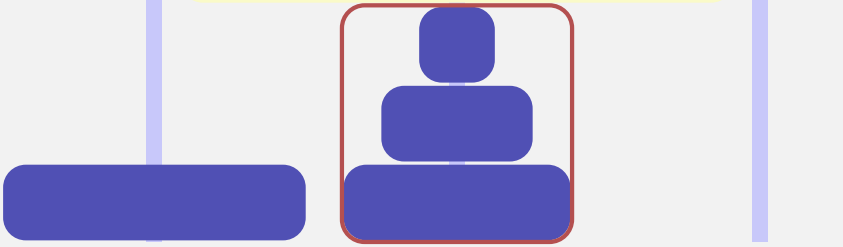


Mal *angenommen*, wir wüssten wie man ...

Links Mitte Rechts

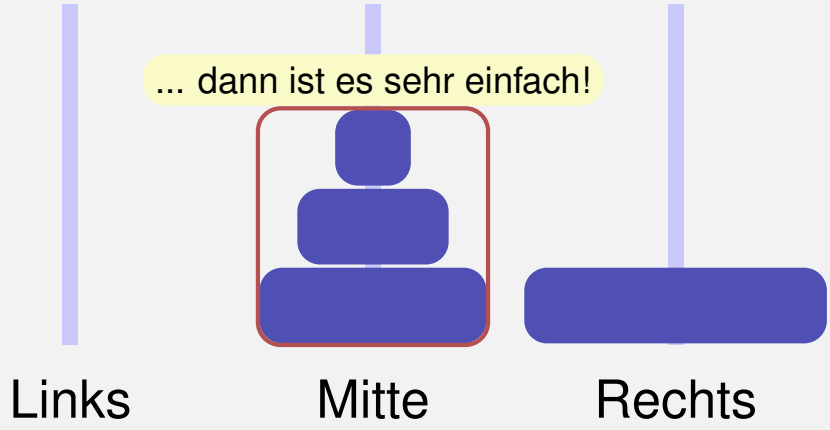
Die Türme von Hanoi - Rekursiver Lösungsansatz

... *drei* Scheiben von einem Stapel zum anderen bringt ...

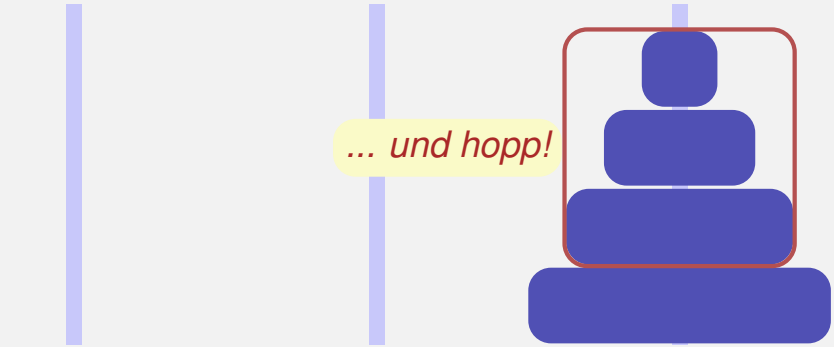


Links Mitte Rechts

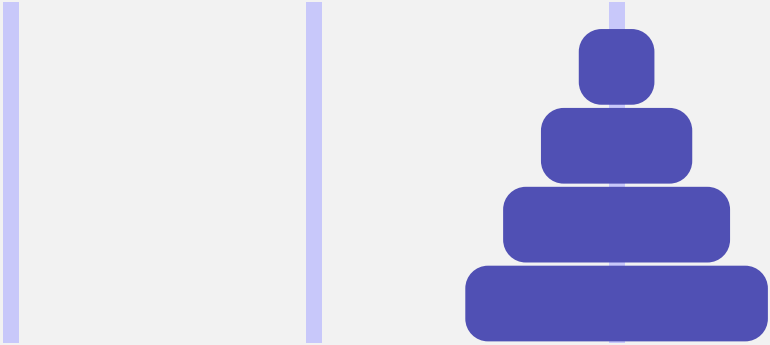
Die Türme von Hanoi - Rekursiver Lösungsansatz



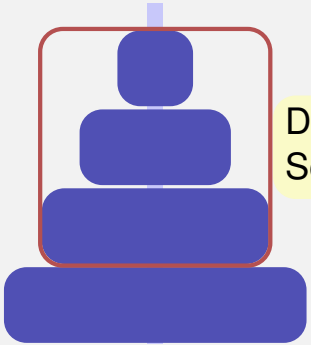
Die Türme von Hanoi - Rekursiver Lösungsansatz



Die Türme von Hanoi - Rekursiver Lösungsansatz



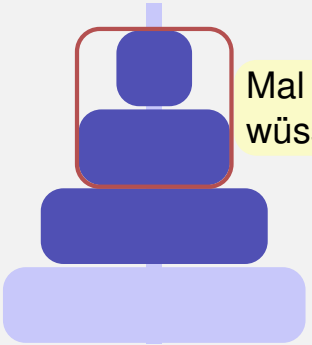
Die Türme von Hanoi - Rekursiver Lösungsansatz



Doch *wie* können wir drei Scheiben bewegen?

Links Mitte Rechts

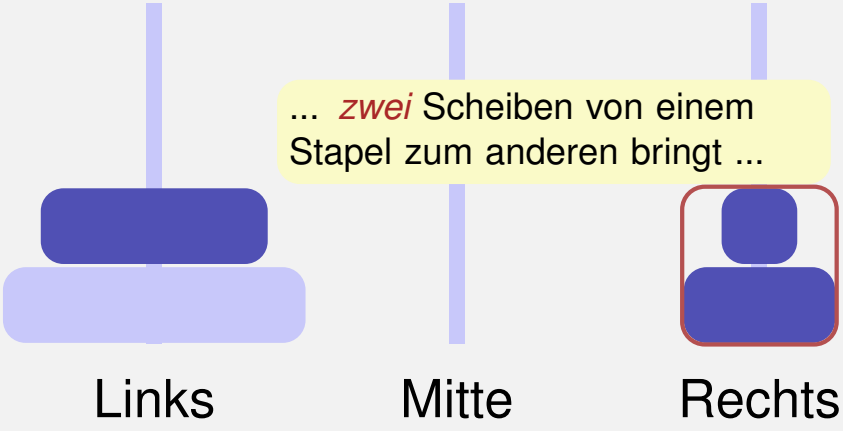
Die Türme von Hanoi - Rekursiver Lösungsansatz



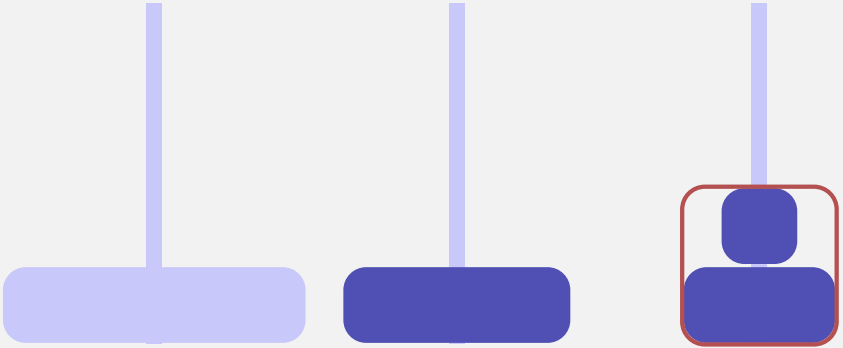
Mal *angenommen*, wir wüssten wie man ...

Links Mitte Rechts

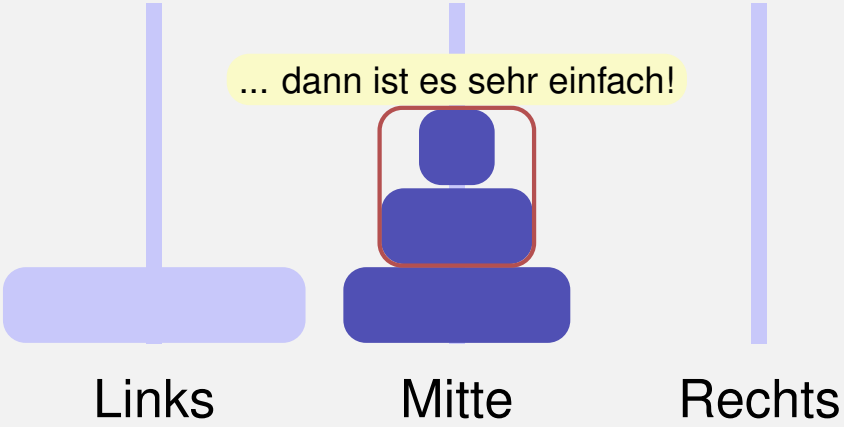
Die Türme von Hanoi - Rekursiver Lösungsansatz



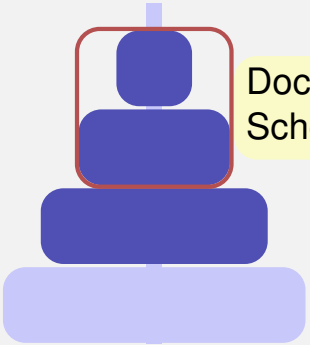
Die Türme von Hanoi - Rekursiver Lösungsansatz



Die Türme von Hanoi - Rekursiver Lösungsansatz



Die Türme von Hanoi - Rekursiver Lösungsansatz

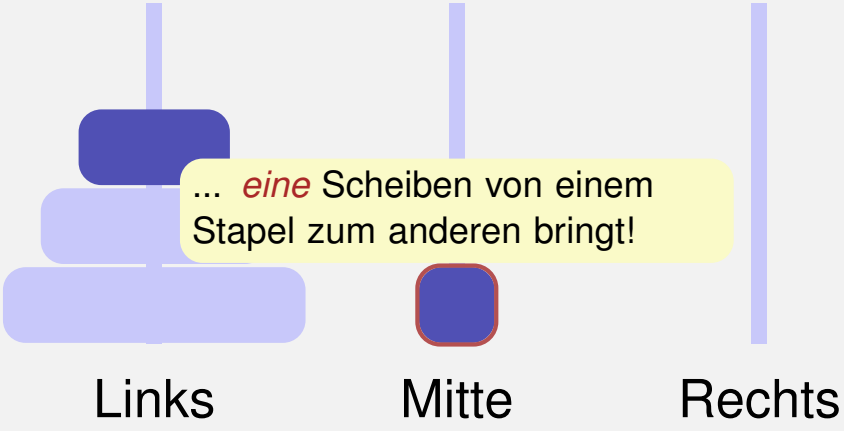


Doch *wie* können wir zwei Scheiben bewegen?

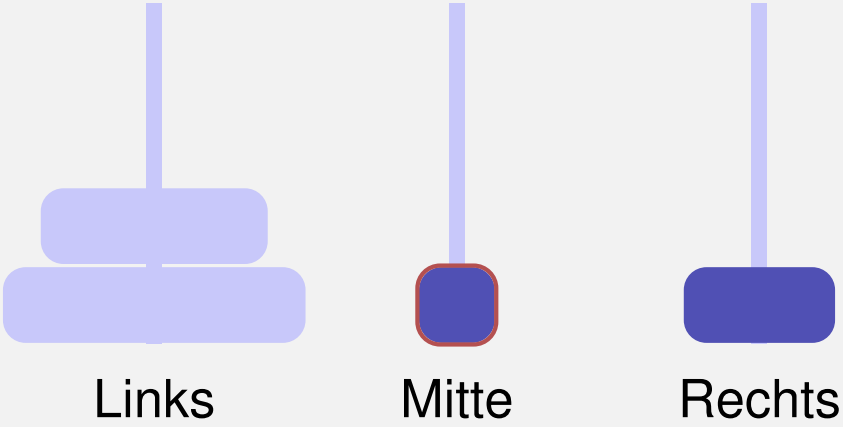
Links Mitte Rechts

Die Türme von Hanoi - Rekursiver Lösungsansatz

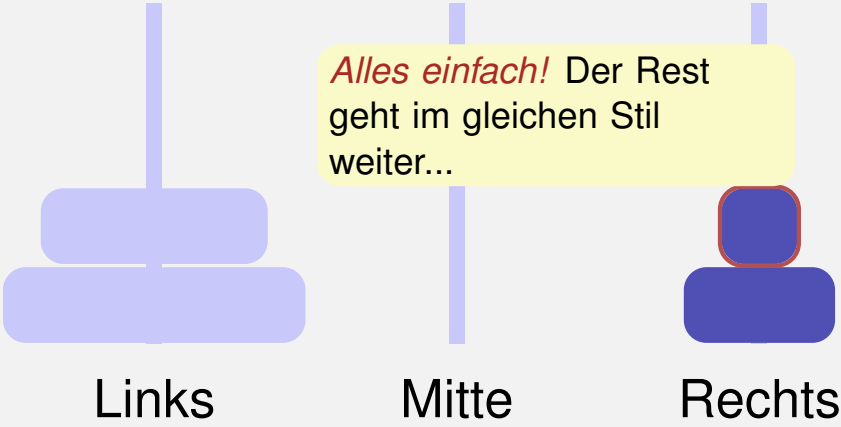
Die Türme von Hanoi - Rekursiver Lösungsansatz



Die Türme von Hanoi - Rekursiver Lösungsansatz



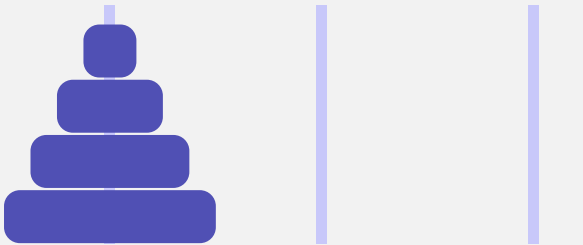
Die Türme von Hanoi - Rekursiver Lösungsansatz



Aufgabe Towers of Hanoi

- Öffnet "Towers of Hanoi" auf **code** expert
- Überlegt euch, wie ihr das Problem mit Stift und Papier angehen würdet
- Programmiert eine Lösung (optional in Gruppen)

Die Türme von Hanoi - Code



Bewege 4 Scheiben von `left` nach `right` mit Hilfsstapel `middle`:

```
move(4,"left","middle","right")
```

Die Türme von Hanoi - Code

move(n**,** src**,** aux**,** dst**)** $\Rightarrow$

1. Bewege die obersten $n-1$ Scheiben von src nach aux mit Hilfsstapel dst:
   move($n-1$**,** src**,** dst**,** aux**);**
2. Bewege 1 Scheibe von src nach dst
   move(1, src**,** aux**,** dst**);**
3. Bewege die obersten $n-1$ Scheiben von aux nach dst mit Hilfsstapel src:
   move($n-1$**,** aux**,** src**,** dst**);**

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case



    }
}
```

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);


    }
}
```

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);
        move(1, src, aux, dst);

    }
}
```

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);
        move(1, src, aux, dst);
        move(n-1, aux, src, dst);
    }
}
```

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case ('move' the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);
        move(1, src, aux, dst);
        move(n-1, aux, src, dst);
    }
}
int main() {
    move(4, "left", "middle", "right");
    return 0;
}
```

Die Türme von Hanoi - Code Alternative

```
void move(int n, const string &src, const string &aux, const string &dst){
    // base case
    if (n == 0) return;

    // recursive case
    move(n-1, src, dst, aux);
    std::cout << src << " --> " << dst << "\n";
    move(n-1, aux, src, dst);
}


int main() {
    move(4, "left", "middle", "right");
    return 0;
}
```

Fragen/Unklarheiten?

Videoempfehlungen

Versucht insbesondere das Konzept von *Recursive Leap of Faith* nachzuvollziehen. Das ist quasi die Induktionsannahme bei einem Induktionsbeweis aus der Mathematik

Videos zum Thema Rekursion

- 5 Simple Steps for Solving Any Recursive Problem
- Towers of Hanoi: A Complete Recursive Visualization

7. Feedback

Euer Feedback an mich

Feedback-Formular

(Nehmt euch Zeit und seid ehrlich)

8. Alte Prüfungsfragen

Prüfungsfrage F^*

Sei F^* das folgende normalisierte Fliesskommazahlensystem[1]

$$F^*(\beta = 2, p = 3, e_{\text{min}} = -1, e_{\text{max}} = 4)$$

Richtig oder Falsch?

1. "1.25 kann im Fliesskommazahlensystem exakt dargestellt werden"
   Richtig, nämlich $1.01 \cdot 2^0$
2. "Es existiert keine Zahl $Z \in F^*$, für die gilt: $0.0625 < Z < 0.25$"
   Richtig, die kleinste darstellbare Zahl ist 0.5 (i.e., $1.0 * 2^{-1}$)
3. "3.25 kann genau in F^* dargestellt werden"
   Falsch, $3.25 = 1,101 * 2^1$ würde die Genauigkeit $p \geq 4$ erfordern

[1] Erinnerung: die Genauigkeit (Anzahl der Ziffern) schliesst das führende Bit ein.

Prüfungsfrage "Schleife"

```
int sum = 17;
int i = 1;

do {
    i += sum;
    sum = sum / 2;
} while (i > sum && sum >= 0);

std::cout << sum;
```

Welche Aussage beschreibt den Output am besten?

- 17
- 8
- Terminiert nie
- 18

Prüfungsfrage "Loop Termination"

```
int sum = 17;
int i = 1;

do {
    i += sum;
    sum = sum / 2;
} while (i > sum && sum >= 0);

std::cout << sum;
```

Antwort: Es terminiert nie!

- Division von zwei positiven `int` kann nicht negativ sein
  $\Rightarrow$ `sum >= 0` ist immer wahr!
- Nach der ersten Ausführung des do-Blocks: `i > sum`.
  `sum` ist monoton fallend, `i` ist monoton steigend
  $\Rightarrow$ `i > sum` ist immer wahr.

9. Outro

Allgemeine Fragen?

Bis zum nächsten Mal

Schöne Woche noch!