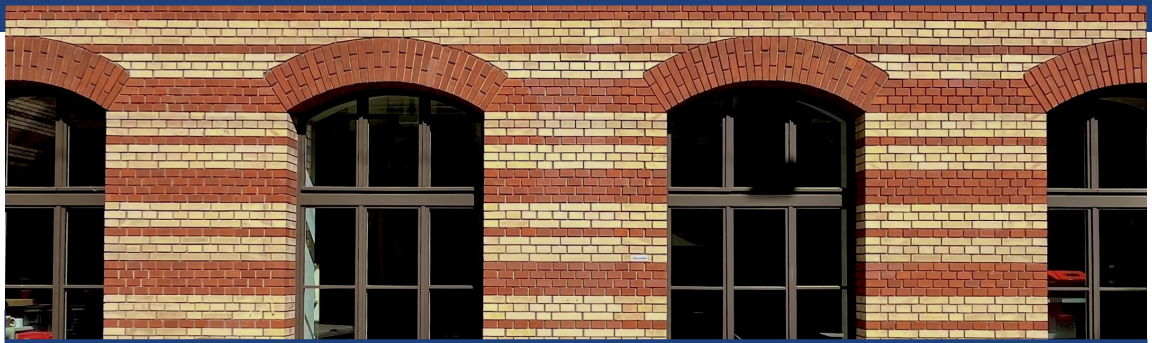


Exercise Session — Computer Science — 08

Recursion, Structs

Overview

Today's Plan

Structs
Recursion
- Exercise "Power Set"
- Exercise "The Towers of Hanoi"

n.ethz.ch/~iopopa

Link to Webpage

Send an e-Mail

1. Feedback regarding **code** expert

General things regarding code expert

- Avoid returning vectors in the functions you are writing

```
std::vector<int> read_vector(){
  std::vector<int> v;
  //Reading the vector
}
int main(){
  std::vector<int> v = read_vector();
}
```

- The vector is returned by value, which means it's copied when returned to main

General things regarding code expert

- Pass the vector by reference in the function instead

```
void read_vector(std::vector<int>& v){
  //Reading the vector elements directly in v
}
int main(){
  std::vector<int> v;
  read_vector();
}
```

Exercise 7: Task 1: Const and reference types

- You are allowed to return a const reference to a non-const variable in C++

```
const int& foo(int& i) {
  return ++i;
}
```

Exercise 7: Task 1: Const and reference types

- You are allowed to return a const reference to a non-const variable in C++

```
const int& foo(int& i) {
  return ++i;
}
```

- But you cannot change the value of `i` through the pointer that is returned by the function due to constness
- For example, writing something like: `foo(i) = 8;` or `foo(i)++` will produce a runtime error because you should not atttempt to change constant variables

Important changes regarding feedback

- Due to high workload, from today onwards you will only receive feedback on request (unless I see something fundamentally wrong)
- This "request" can look like this and should be placed at the very top of the code:

```
// FEEDBACK PLEASE
// - especially regarding lines 12, 13 and 42
// QUESTIONS
// - [re: line 42] I was wondering if [...]
```

- TA points are still being provided
- If the XP has to be set to 0 somewhere, I will mention in the feedback why

Any questions regarding **code** expert on your part?

2. Structs

Example for Structs

Example for Structs

```
struct strange {
    int n;
    bool b;
    std::vector<int> a = std::vector<int>(0);
};

int main () {
    strange x = {1, true, {1,2,3}};
    strange y = x;                                  // all elements are copied
    std::cout << y.n << " " << y.a[2] << "\n";  // outputs: 1 3
    return 0;
}
```

Exercise "Geometry Exercise"

- Open "Geometry Exercise" on **code** expert

Exercise "Geometry Exercise"

- Open "Geometry Exercise" on **code** expert
- Think about how you would approach the problem with pen and paper

Exercise "Geometry Exercise"

- Open "Geometry Exercise" on **code** expert
- Think about how you would approach the problem with pen and paper
- Group Programming time!

3. Recursion

3. Recursion

3.1. Exercise "Power Set"

Exercise "Power Set"

Recap

- A power set is the set of all subsets

Exercise "Power Set"

Recap

- A power set is the set of all subsets

$$\mathcal{P}(S) := \{X \mid X \subseteq S\}$$

Exercise "Power Set"

Recap

- A power set is the set of all subsets

$$\mathcal{P}(S) := \{X \mid X \subseteq S\}$$

- Example:
  - Given the set $A = \{a, b, c\}$

Exercise "Power Set"

Recap

- A power set is the set of all subsets

$$\mathcal{P}(S) := \{X \mid X \subseteq S\}$$

- Example:
  - Given the set $A = \{a, b, c\}$
  - Its power set is $\mathcal{P}(A) = \{\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$

Primer on `set.h`

- `set` is a self-made type! (a `class`)
- How does it work? See for yourself in `set.h`!

```
template <typename T>
class Set {
    public:
    Set(const Set& other);
    // Creates an empty set
    Set();
    // Creates a new set from a set of elements
    Set(const std::set<T>& elements);
    // Creates a new set from a single element
    Set(T element);
    // ...
};
```

Exercise Power Set

- Open "Power Set" on **code** expert

Exercise Power Set

- Open "Power Set" on **code** expert
- Think about how you would approach the problem with pen and paper

Exercise Power Set

- Open "Power Set" on **code** expert
- Think about how you would approach the problem with pen and paper
- Implement a solution (optionally in groups)

Exercise Power Set

- Open "Power Set" on **code** expert
- Think about how you would approach the problem with pen and paper
- Implement a solution (optionally in groups)
- You can find the functionalities of the type `set` in the `main.cpp` file
- Possible key questions: For which (simple) cases do we always know the solution? Is there a pattern that the power sets follow when another element is added?

Solution to "Power Set" (Conceptually)

Given: $\{a, b, c, d\}$

[1]Here is where the *Recursive Leap of Faith* kicks in

Solution to "Power Set" (Conceptually)

Given: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

[1] Here is where the *Recursive Leap of Faith* kicks in

Solution to "Power Set" (Conceptually)

Given: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

[1] Here is where the *Recursive Leap of Faith* kicks in

Solution to "Power Set" (Conceptually)

Given: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

```
// get power set for remaining subset
```
[1]

[1] Here is where the *Recursive Leap of Faith* kicks in

Solution to "Power Set" (Conceptually)

Given: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

```
// get power set for remaining subset[1]
```

$$\mathcal{P}(\{b, c, d\}) = \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

[1] Here is where the *Recursive Leap of Faith* kicks in

Solution to "Power Set" (Conceptually)

Given: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

```
// get power set for remaining subset[1]
```

$$\mathcal{P}(\{b, c, d\}) = \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

```
// init result with power set of remaining subset
```

[1] Here is where the *Recursive Leap of Faith* kicks in

Solution to "Power Set" (Conceptually)

Given: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

```
// get power set for remaining subset[1]
```

$$\mathcal{P}(\{b, c, d\}) = \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

```
// init result with power set of remaining subset
```

$$\texttt{result} \leftarrow \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

[1] Here is where the *Recursive Leap of Faith* kicks in

Solution to "Power Set" (Conceptually)

Given: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

```
// get power set for remaining subset[1]
```

$$\mathcal{P}(\{b, c, d\}) = \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

```
// init result with power set of remaining subset
```

$$\texttt{result} \leftarrow \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

```
// add first element to every set in the powerset
```

[1] Here is where the *Recursive Leap of Faith* kicks in

Solution to "Power Set" (Conceptually)

Given: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

```
// get power set for remaining subset[1]
```

$$\mathcal{P}(\{b, c, d\}) = \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

```
// init result with power set of remaining subset
```

$$\texttt{result} \leftarrow \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

```
// add first element to every set in the powerset
```

$$\left\{ \begin{matrix} \{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots, \\ \{a\}, \{a, b\}, \{a, c\}, \{a, d\}, \{a, b, c\}, \dots, \end{matrix} \right\}$$

[1] Here is where the *Recursive Leap of Faith* kicks in

Solution to "Power Set" (Base case)

Solution to "Power Set" (Base case)

```
SetOfCharSets power_set(const CharSet& set) {
  // base case: empty set
  if (set.size() == 0) {
    return SetOfCharSets(CharSet());
  }
```

Solution to "Power Set"

```
// set has at least 1 element -> split set into two sets.
CharSet first_element_subset = CharSet(set.at(0));
CharSet remaining_subset = set - first_element_subset;

// get power set for remaining subset
SetOfCharSets remaining_subset_power_set = power_set(remaining_subset);

// init result with power set of remaining subset
SetOfCharSets result = remaining_subset_power_set;

// add first element to every set in the powerset
for (unsigned int i = 0; i < remaining_subset_power_set.size(); ++i) {
  result.insert(first_element_subset + remaining_subset_power_set.at(i));
}

return result;
```

Questions?

3.2. Exercise "The Towers of Hanoi"

The Towers of Hanoi

Struggling with this exercise is a bit of a rite of passage for newbie programmers. It's notoriously difficult if one is not familiar with recursion.

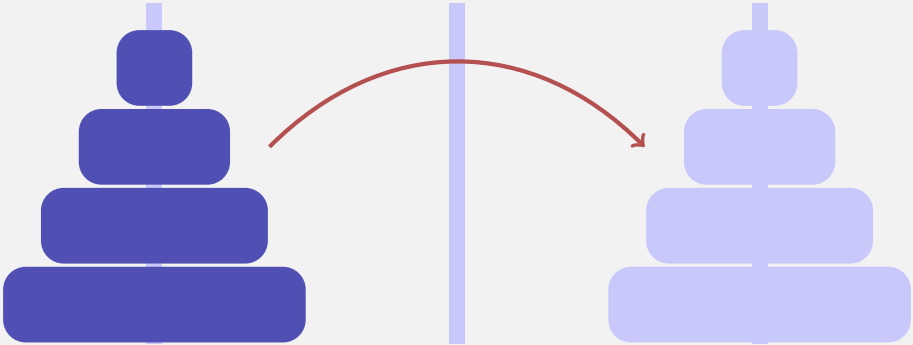
The Towers of Hanoi

Struggling with this exercise is a bit of a rite of passage for newbie programmers. It's notoriously difficult if one is not familiar with recursion.

Experiment: The Towers of Hanoi

left middle right

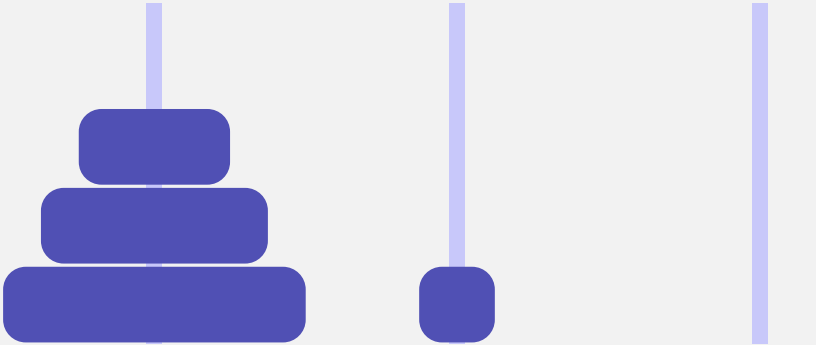
Experiment: The Towers of Hanoi



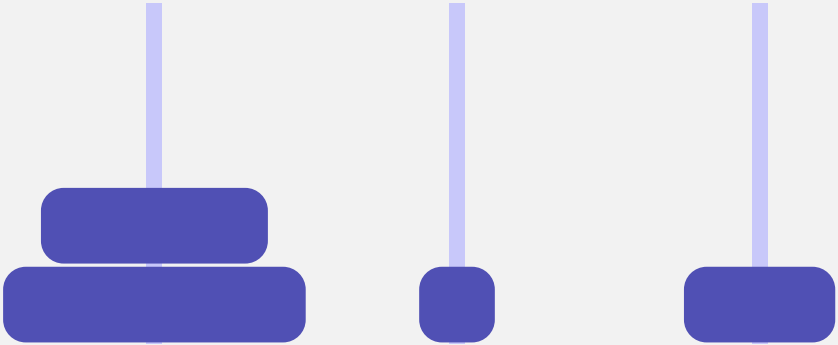
Die Türme von Hanoi - So gehts!

left middle right

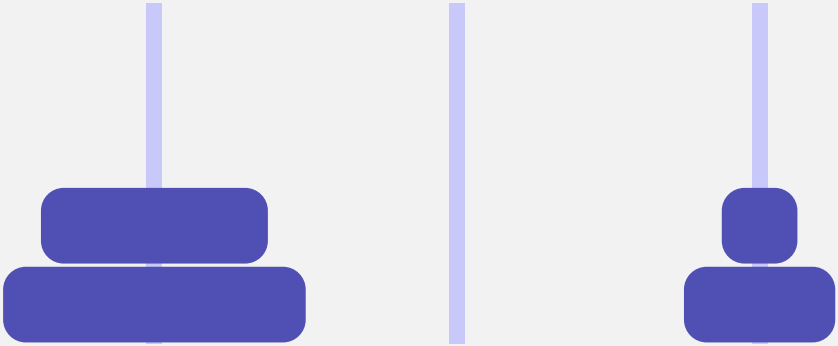
Die Türme von Hanoi - So gehts!



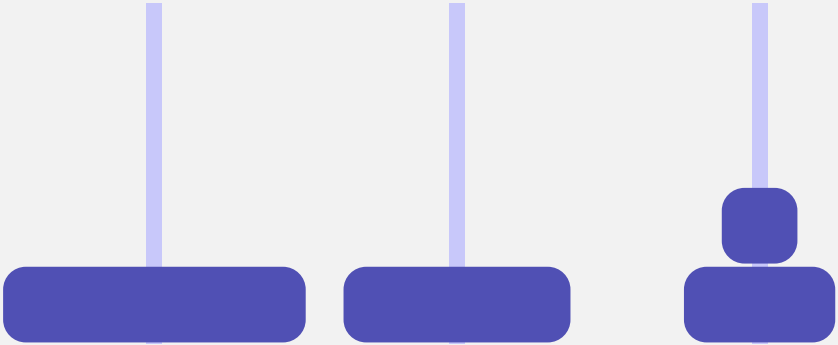
Die Türme von Hanoi - So gehts!



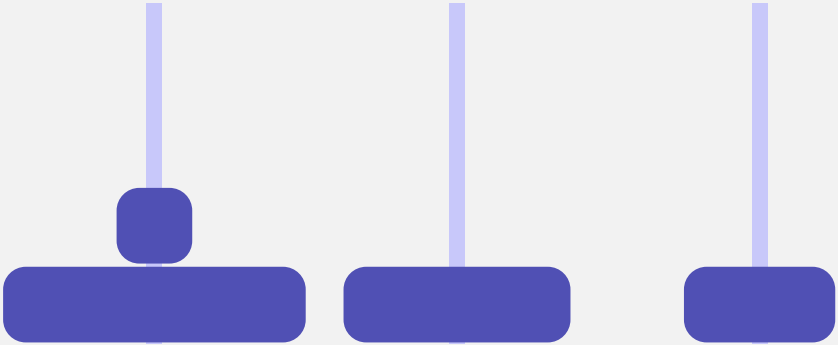
Die Türme von Hanoi - So gehts!



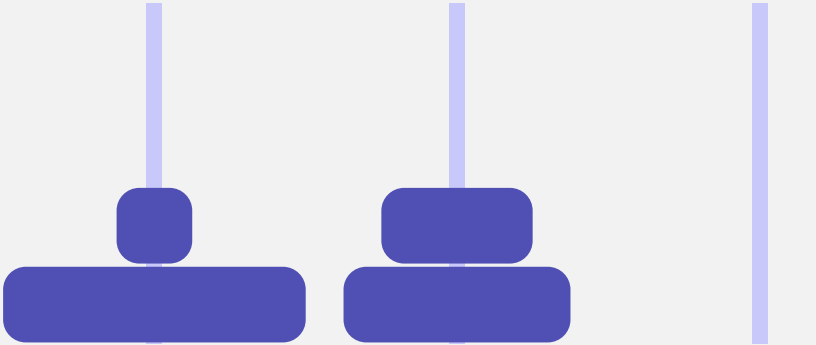
Die Türme von Hanoi - So gehts!



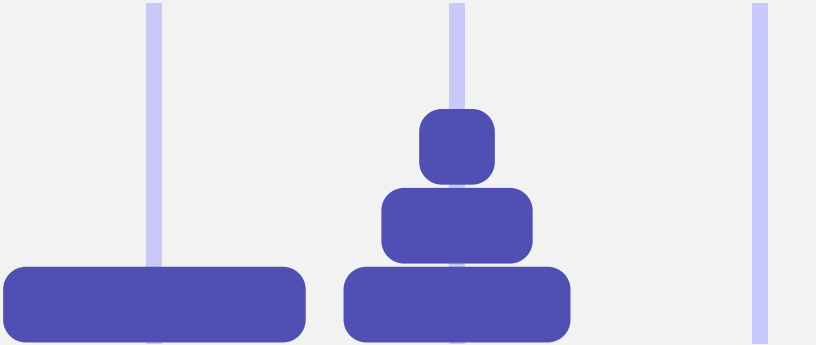
Die Türme von Hanoi - So gehts!



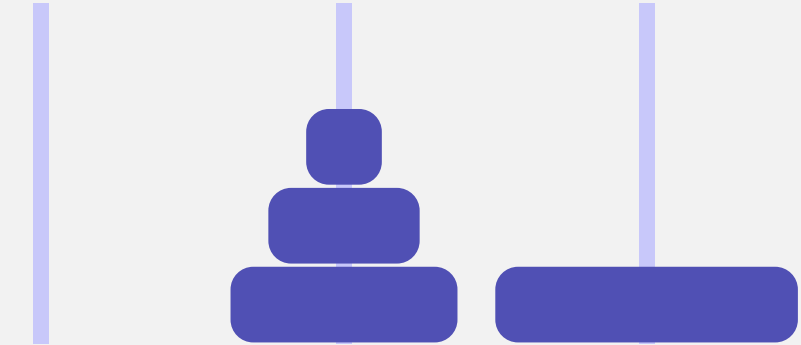
Die Türme von Hanoi - So gehts!



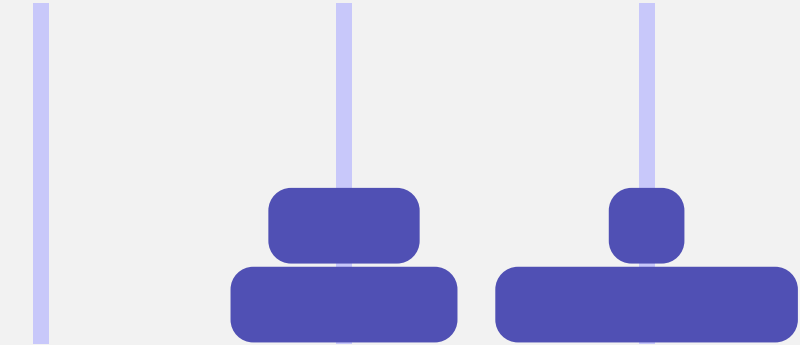
Die Türme von Hanoi - So gehts!



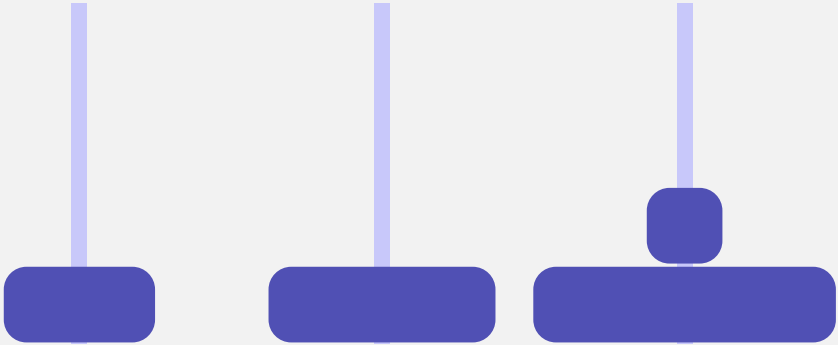
Die Türme von Hanoi - So gehts!



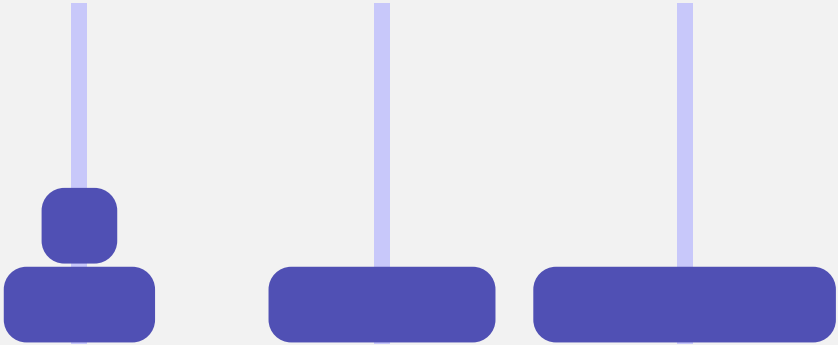
Die Türme von Hanoi - So gehts!



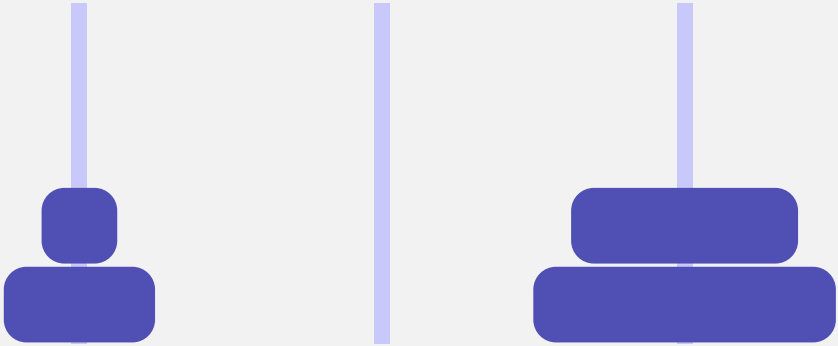
Die Türme von Hanoi - So gehts!



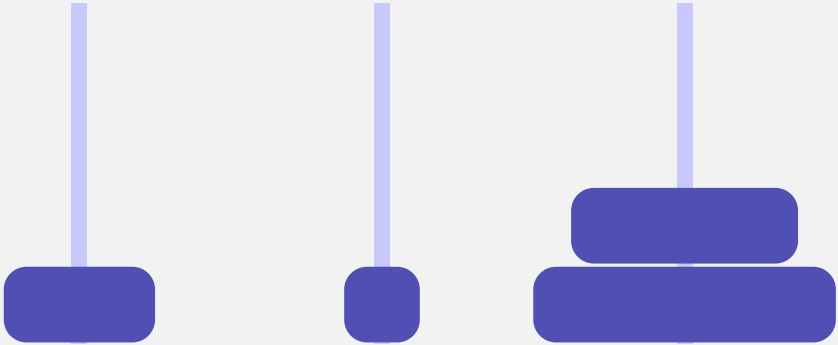
Die Türme von Hanoi - So gehts!



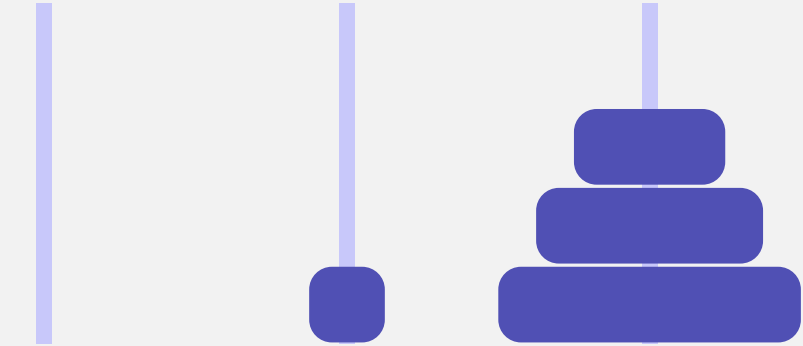
Die Türme von Hanoi - So gehts!



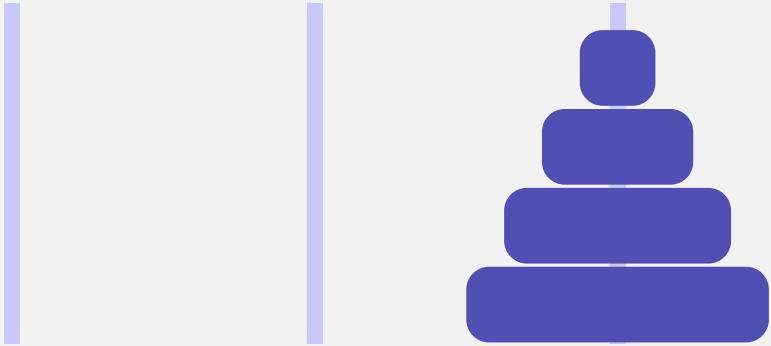
Die Türme von Hanoi - So gehts!



Die Türme von Hanoi - So gehts!



Die Türme von Hanoi - So gehts!

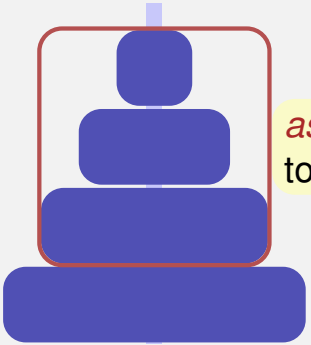


left middle right

The Towers of Hanoi – Recursive Approach

left middle right

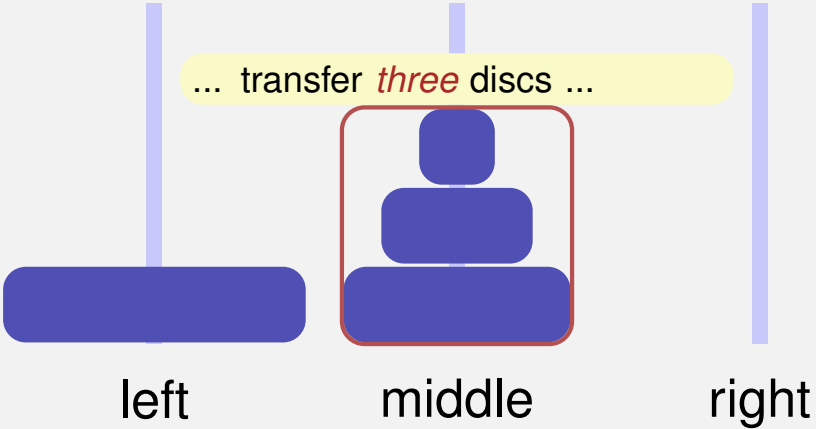
The Towers of Hanoi – Recursive Approach



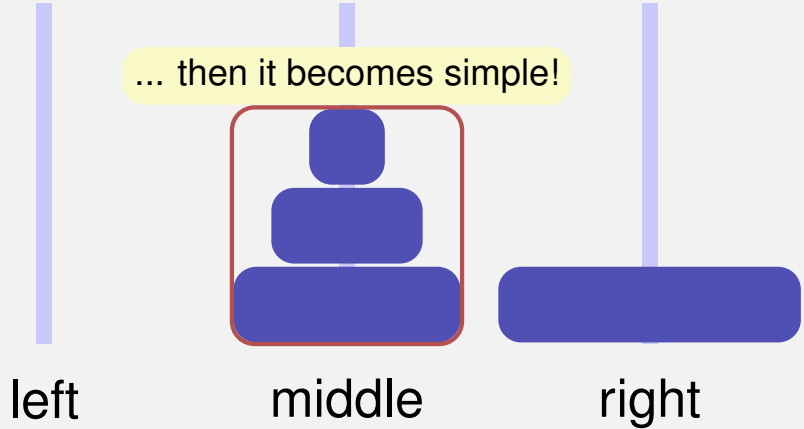
assume we knew how to ...

left middle right

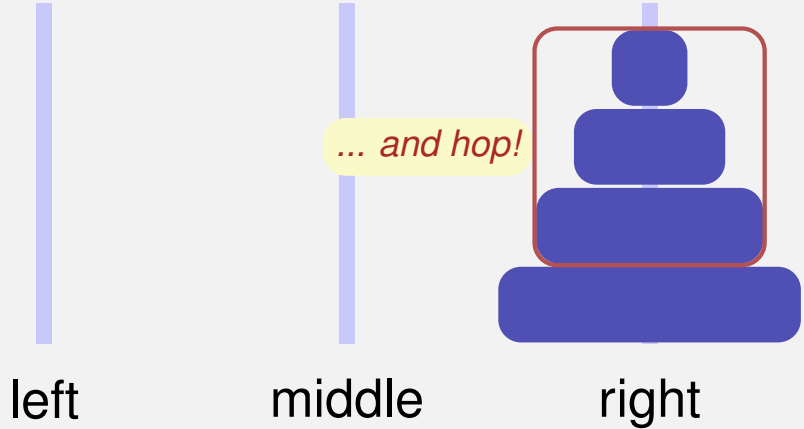
The Towers of Hanoi – Recursive Approach



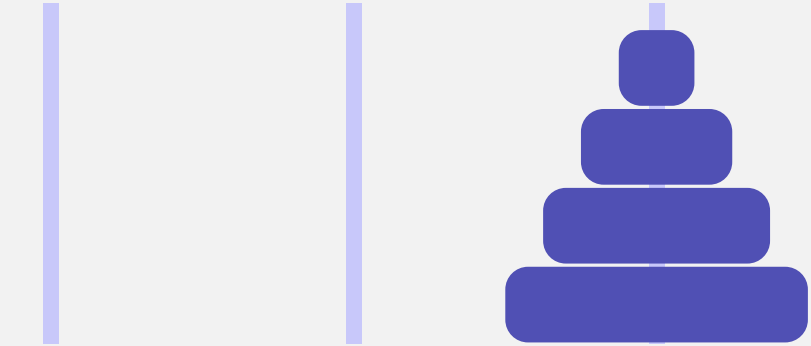
The Towers of Hanoi – Recursive Approach



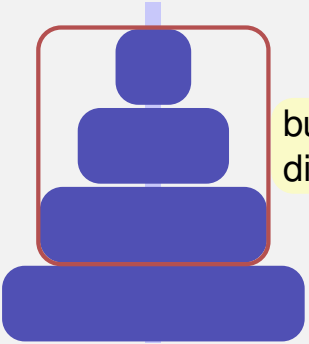
The Towers of Hanoi – Recursive Approach



The Towers of Hanoi – Recursive Approach



The Towers of Hanoi – Recursive Approach



but *how* can we move three discs?

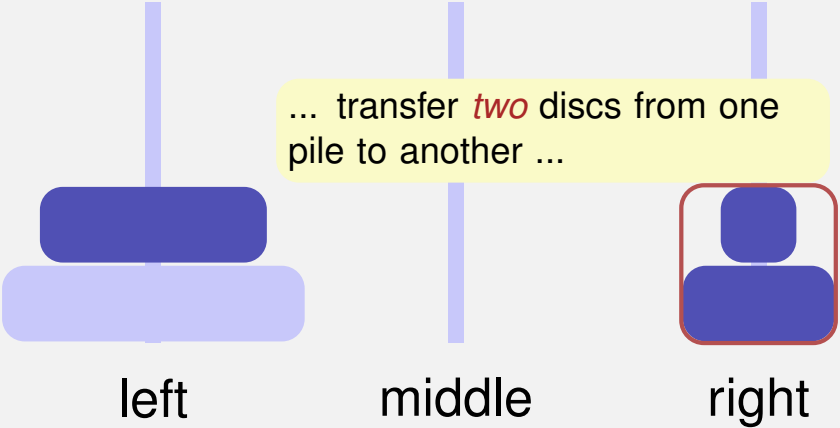
left middle right

The Towers of Hanoi – Recursive Approach

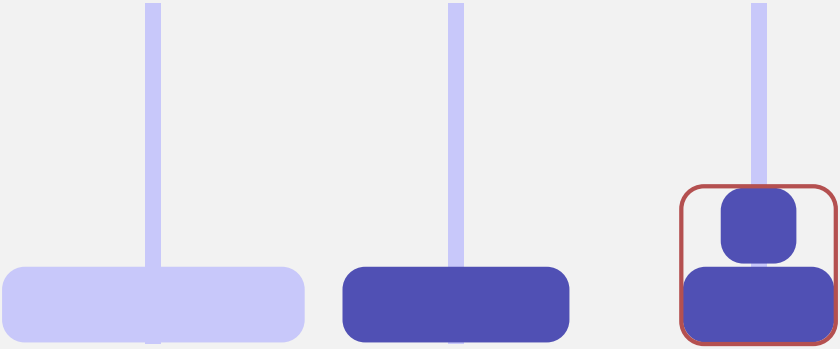
assume we knew how to ...

left middle right

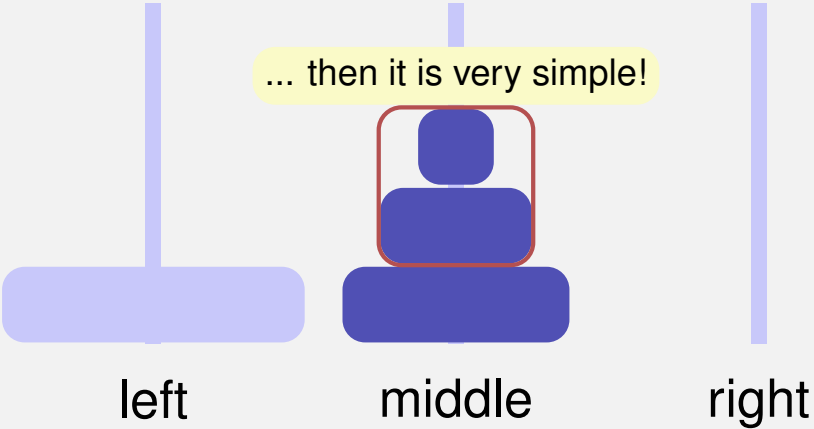
The Towers of Hanoi – Recursive Approach



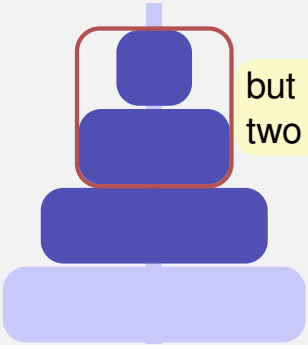
The Towers of Hanoi – Recursive Approach



The Towers of Hanoi – Recursive Approach



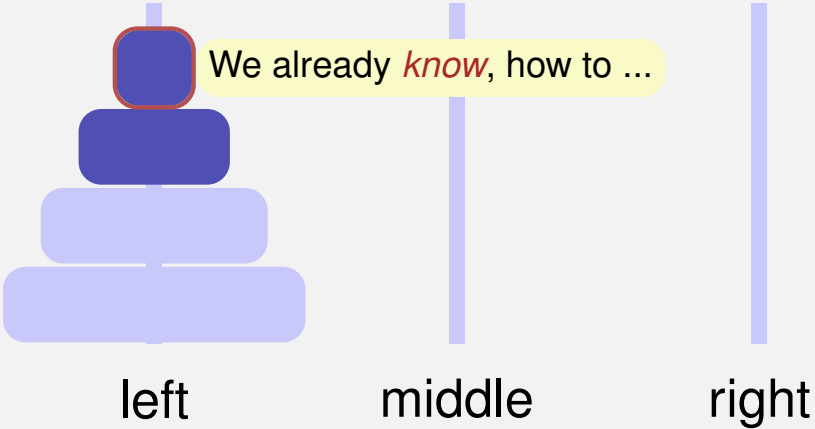
The Towers of Hanoi – Recursive Approach



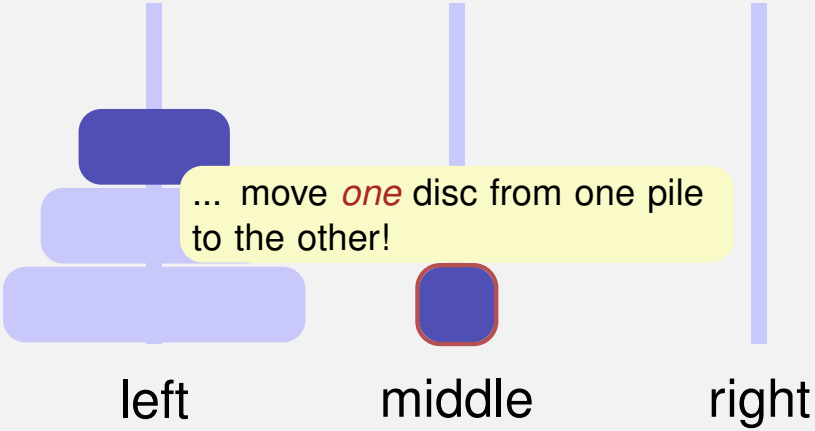
but *how* can we transfer two discs?

left middle right

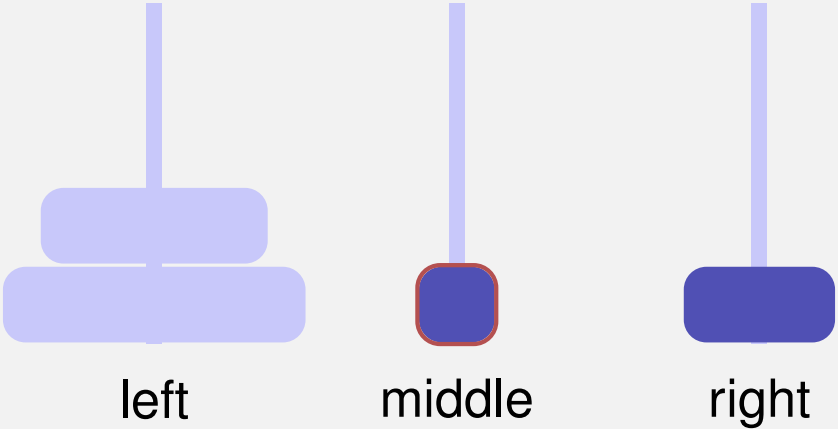
The Towers of Hanoi – Recursive Approach



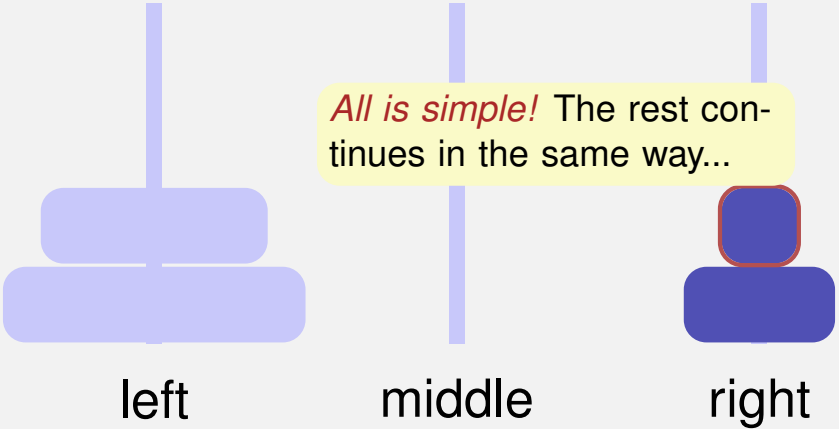
The Towers of Hanoi – Recursive Approach



The Towers of Hanoi – Recursive Approach



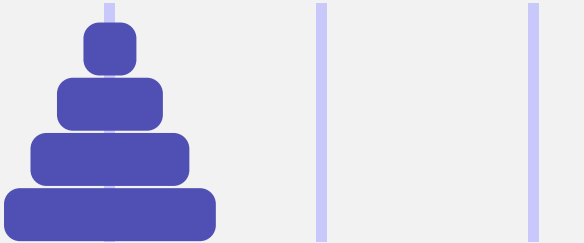
The Towers of Hanoi – Recursive Approach



Exercise Towers of Hanoi

- Open "Towers of Hanoi" on **code** expert
- Think about how you would approach the problem with pen and paper
- Implement a solution (optionally in groups)

The Towers of Hanoi – Code



Move 4 discs from `left` to `right` with auxiliary staple `middle`:

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

The Towers of Hanoi – Code

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

1. Move the top $n-1$ discs from src to aux with auxiliary staple dst:
   `move(`$n-1$`,` src`,` dst`,` aux`);`
2. Move 1 disc from src to dst
   `move(1,` src`,` aux`,` dst`);`
3. Move the top $n-1$ discs from aux to dst with auxiliary staple src:
   `move(`$n-1$`,` aux`,` src`,` dst`);`

The Towers of 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



    }
}
```

The Towers of 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);


    }
}
```

The Towers of 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);

    }
}
```

The Towers of 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);
    }
}
```

The Towers of 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;
}
```

The Towers of 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;
}
```

Questions?

4. Outro

General Questions?

See you next time

Have a nice week!