ETH zürich

Exercise Session W09

Computer Science (CSE & CBB & Statistics) – AS 23

Overview

Today's Agenda

Follow-up
Feedback on **code** expert
Objectives
Recursion II
Structs
Outro

n.ethz.ch/~agavranovic

1. Follow-up

Follow-up from previous exercise sessions

- Efficiency of "`read_matrix`":
  - The version with the pre-initialized matrix should generally be more efficient, since it uses less dynamically allocated memory[1]
  - It only really makes a difference for very big matrices, so you do not need to worry about the efficiency for now

[1] You will learn what this means towards the end of the semester

2. Feedback on **code** expert

General things regarding **code** expert

A few simiplifications for your code[2]

```
if(condition == true){
  // ...
}
```

→

```
if(condition){
  //...
}
```

```
if(condition){
  return true;
} else {
  return false;
}
```

→

```
return condition;
```

In case of function that return a `bool`

[2] Remember: simplifications aren't always better for comprehension

Specific things regarding **code** expert

E7:T1: "Const and reference types"

- What does `const` mean?
  - Once a `const` variable has been initialized, its value cannot be changed
  - The variable can be used in the program (but "read only")
- When is constness (not) respected?
  - Default: if nothing is declared `const` then constness is respected
  - Otherwise: you must not attempt to modify the value of a `const` variable (no "write access")

Questions?

3. Objectives

Objectives

- ☐ be able to solve more advanced problems involving recursion
- ☐ be able to define and use structs

4. Recursion II

Exercise Power Set

DiskMath Recap

- A power set is the set of all subsets

Exercise Power Set

DiskMath Recap

- A power set is the set of all subsets
- $2^S = \{X | X \subseteq S\}$

$S = \{\} = 2^0 = 1$

$\{\{\}\}$

$|2^S| = 2^{|S|}$

Exercise Power Set

DiskMath Recap

- A power set is the set of all subsets
- $2^S := \{X | X \subseteq S\}$
- Example:
  - Given the set $A = \{a, b, c\}$

Exercise Power Set

DiskMath Recap

- A power set is the set of all subsets
- $2^S := \{X | X \subseteq S\}$
- Example:
  - Given the set $A = \{a, b, c\}$
  - Its power set is $2^A = \{\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$

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)

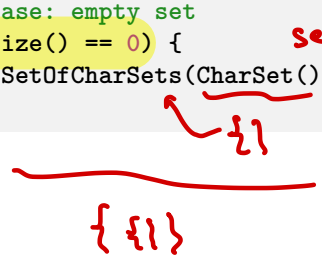
Exercise Power Set

- 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

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;
```

Solution to "Power Set" (Conceptually)

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

[3] 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
```

[3] 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
```

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

[3] 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
```
[3]

[3] 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[3]
```

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

[3] 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[3]
```

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

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

[3] 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[3]
```

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

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

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

[3] 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[3]
```

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

```
// 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
```

[3] 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[3]
```

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

```
// 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
```

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

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

Questions?

Towers of Hanoi

Towers of Hanoi

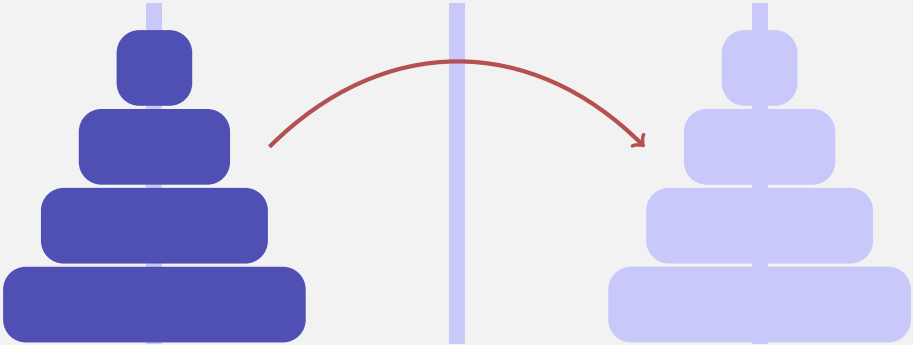
Everyone: it's a game for kids

Programmers:

Experiment: The Towers of Hanoi

left middle right

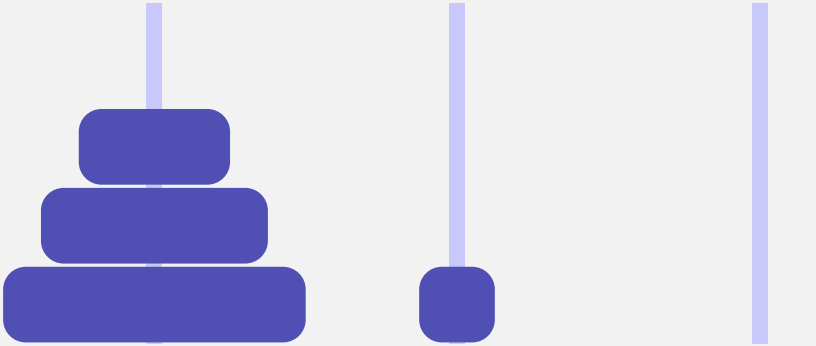
Experiment: The Towers of Hanoi



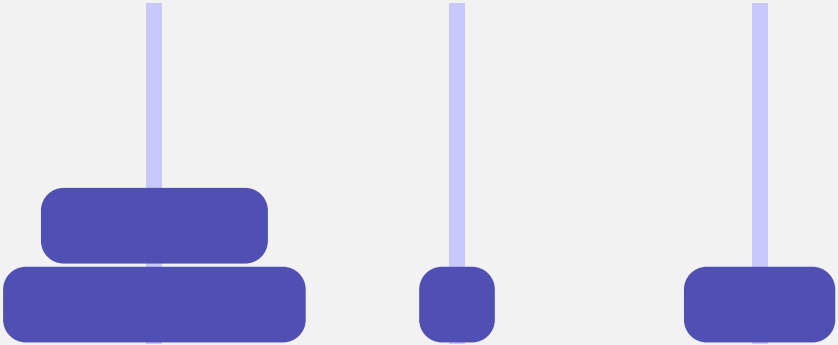
Die Türme von Hanoi - So gehts!

left middle right

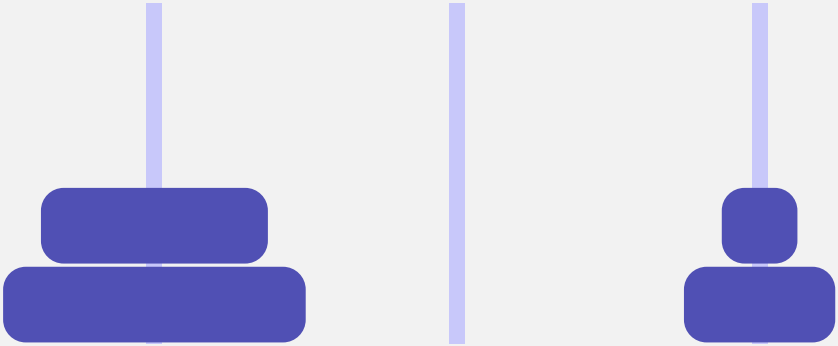
Die Türme von Hanoi - So gehts!



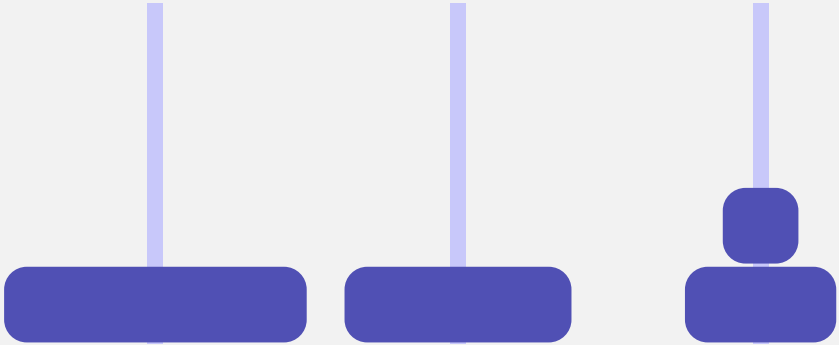
Die Türme von Hanoi - So gehts!



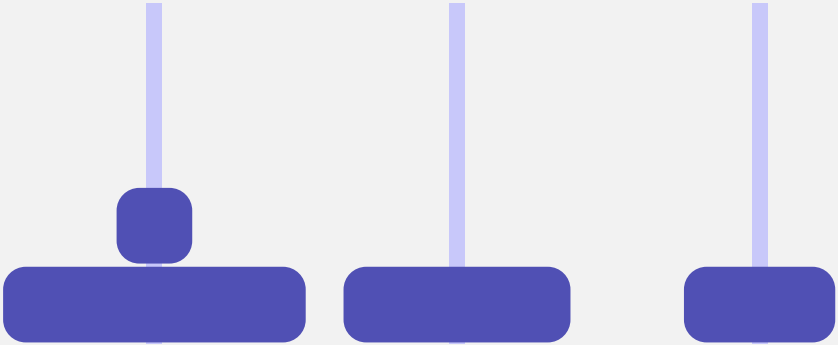
Die Türme von Hanoi - So gehts!



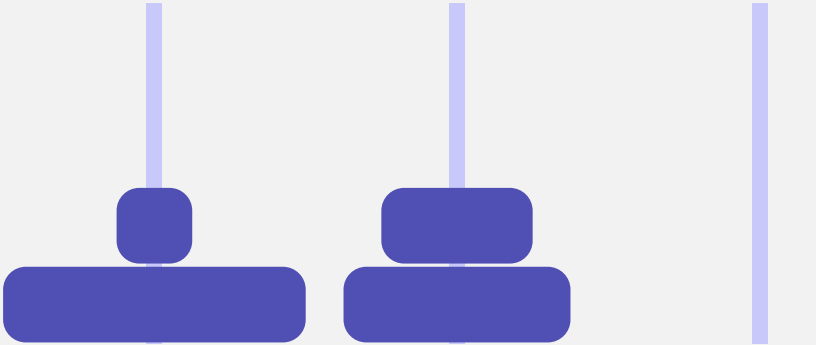
Die Türme von Hanoi - So gehts!



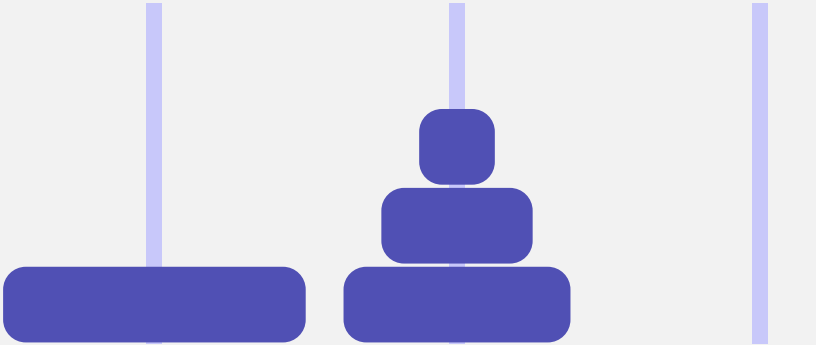
Die Türme von Hanoi - So gehts!



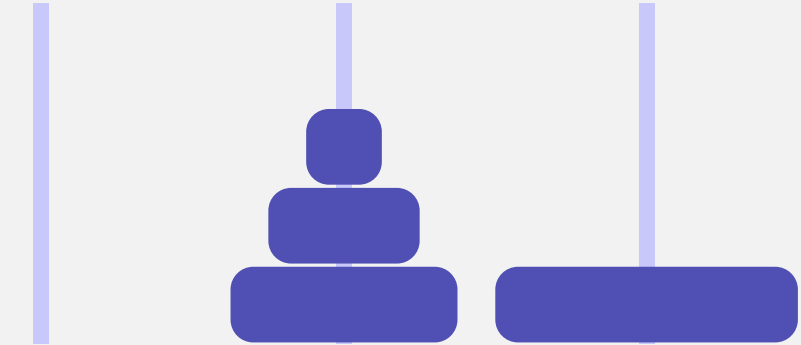
Die Türme von Hanoi - So gehts!



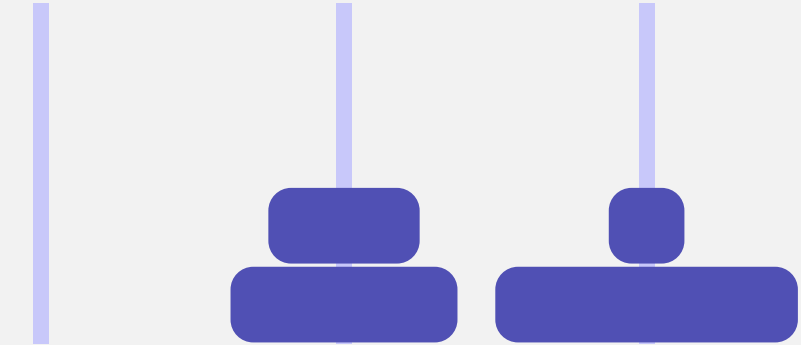
Die Türme von Hanoi - So gehts!



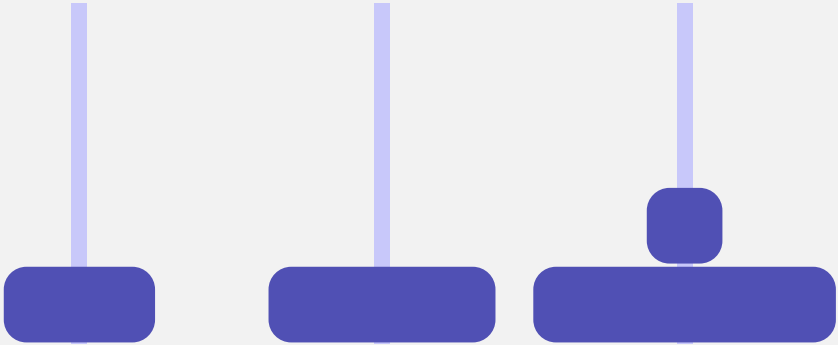
Die Türme von Hanoi - So gehts!



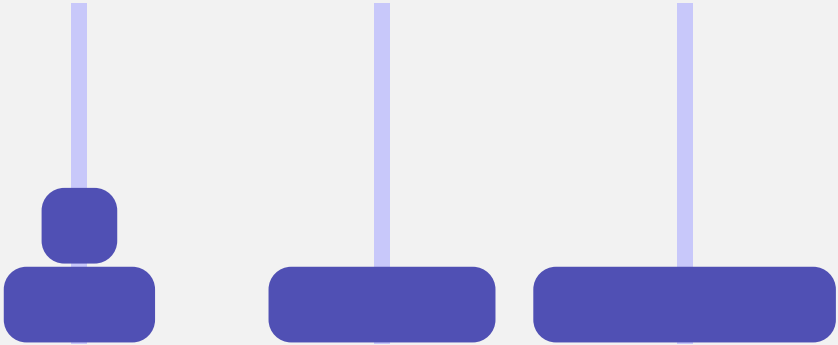
Die Türme von Hanoi - So gehts!



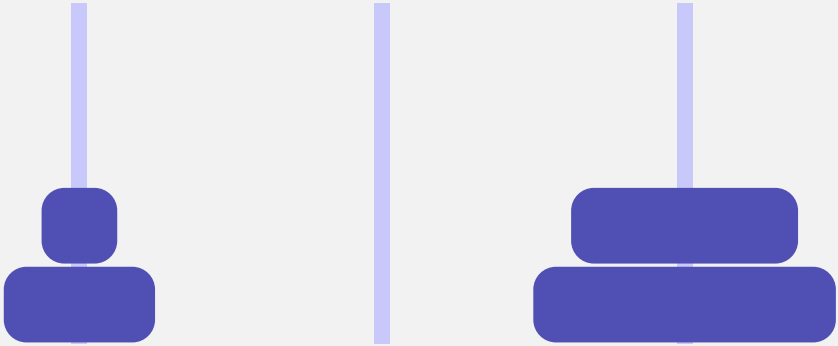
Die Türme von Hanoi - So gehts!



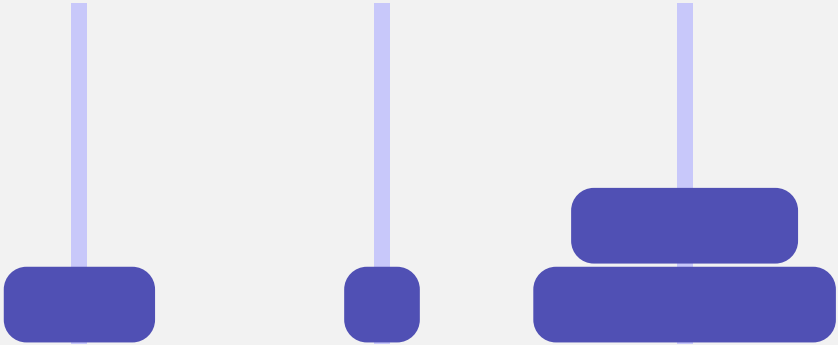
Die Türme von Hanoi - So gehts!



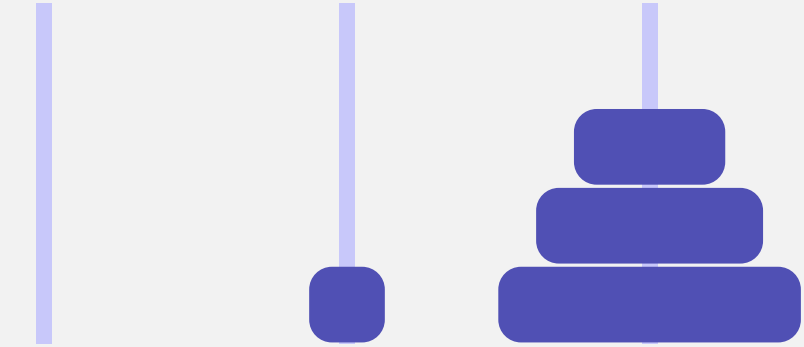
Die Türme von Hanoi - So gehts!



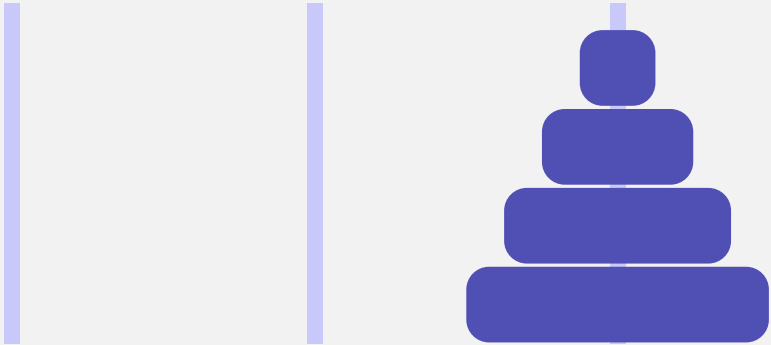
Die Türme von Hanoi - So gehts!



Die Türme von Hanoi - So gehts!



Die Türme von Hanoi - So gehts!



left

middle

right

In the first call!

SRC
source

AUX
auxiliary

DST
destination

Exercise Towers of Hanoi

- Open "Towers of Hanoi" on **code** expert

Exercise Towers of Hanoi

- Open "Towers of Hanoi" on **code** expert
- Think about how you would approach the problem with pen and paper

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)

Exercise Towers of Hanoi

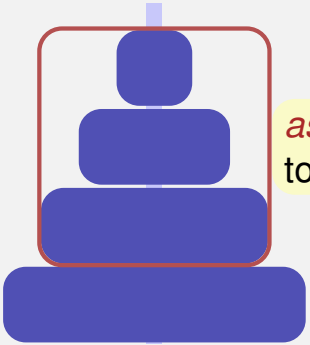
std::string text = "text";

- 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 – Recursive Approach

left middle right

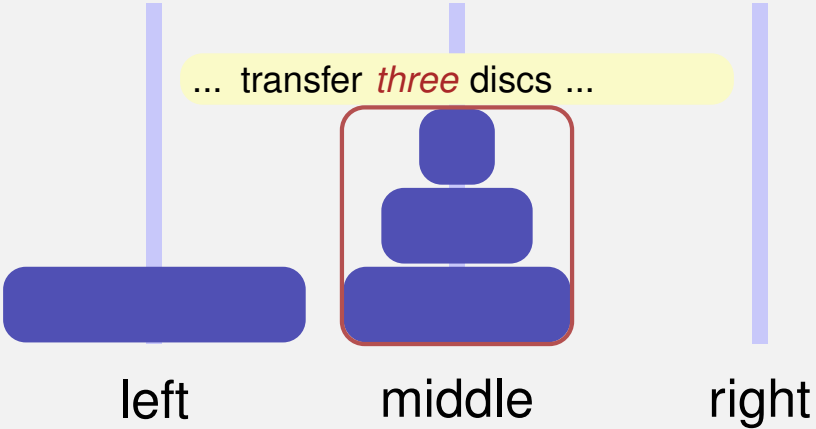
The Towers of Hanoi – Recursive Approach



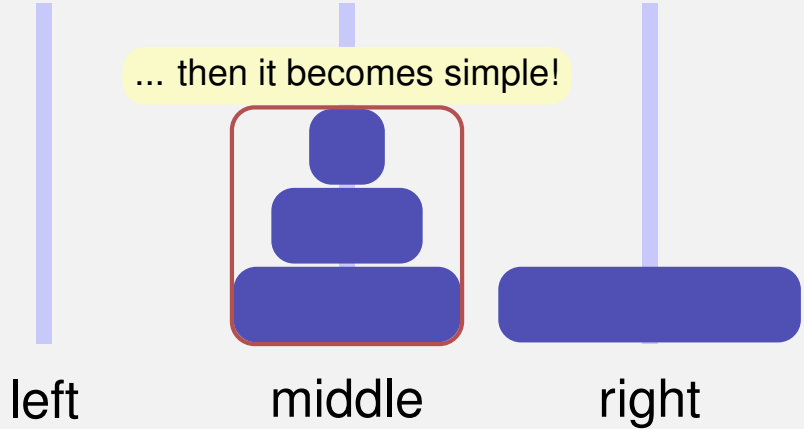
assume we knew how to ...

left middle right

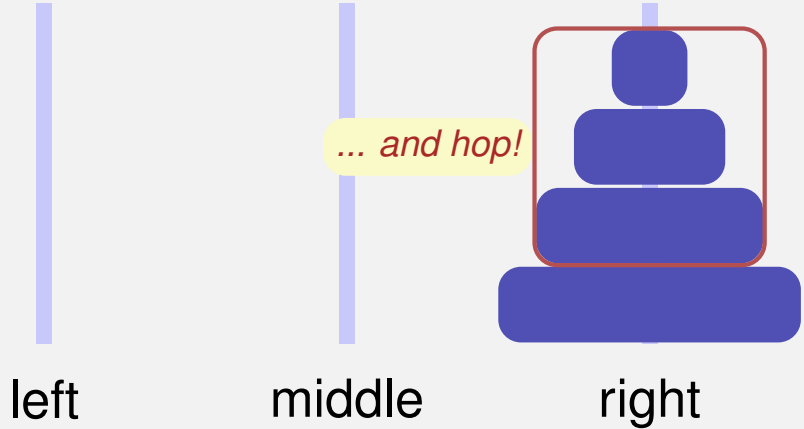
The Towers of Hanoi – Recursive Approach



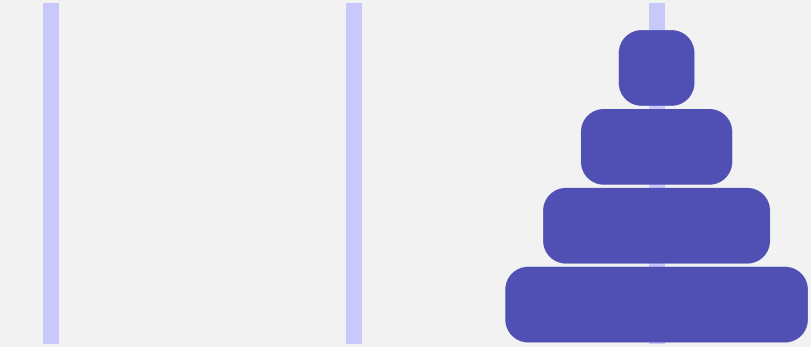
The Towers of Hanoi – Recursive Approach



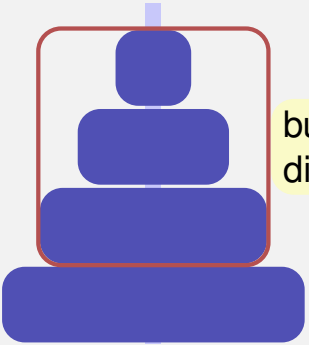
The Towers of Hanoi – Recursive Approach



The Towers of Hanoi – Recursive Approach



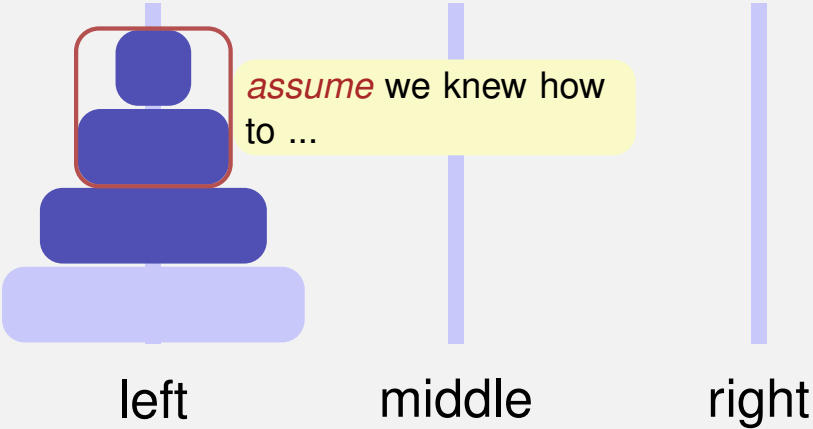
The Towers of Hanoi – Recursive Approach



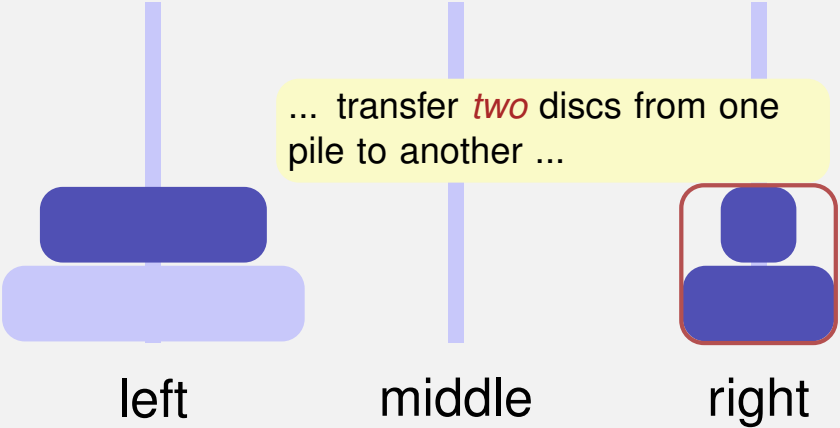
but *how* can we move three discs?

left middle right

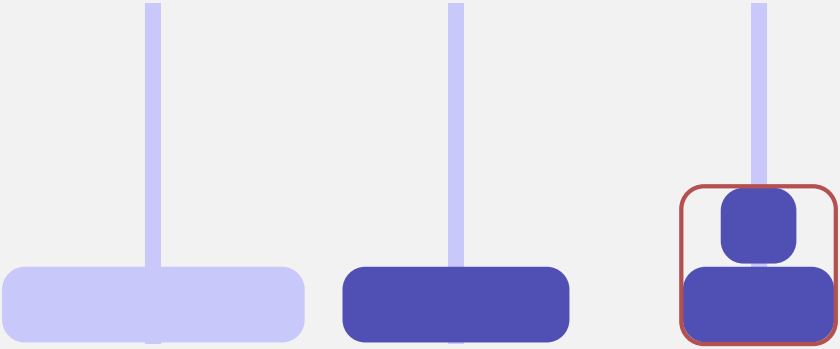
The Towers of Hanoi – Recursive Approach



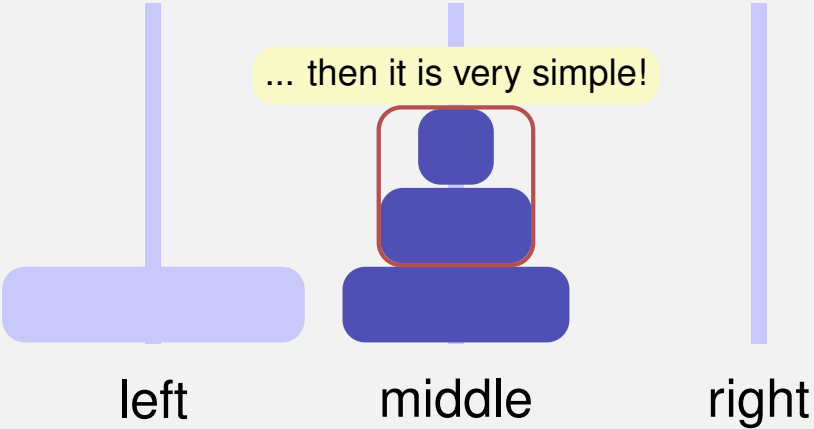
The Towers of Hanoi – Recursive Approach



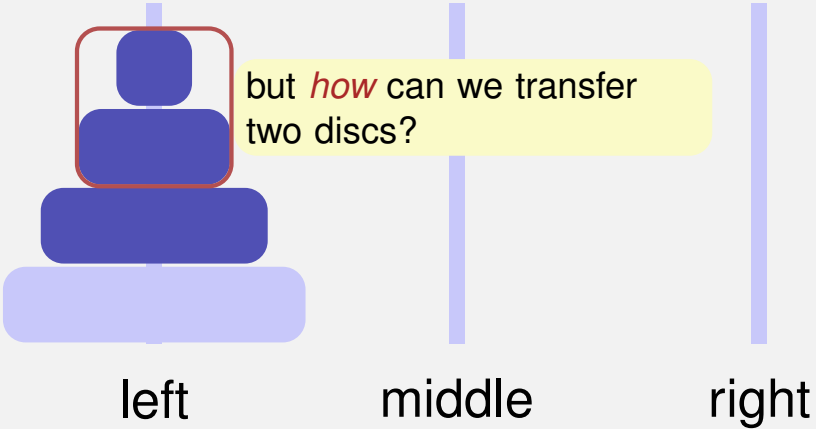
The Towers of Hanoi – Recursive Approach



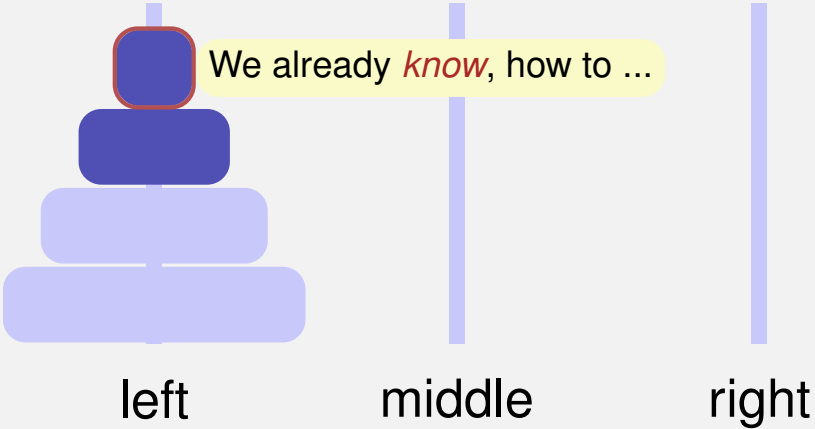
The Towers of Hanoi – Recursive Approach



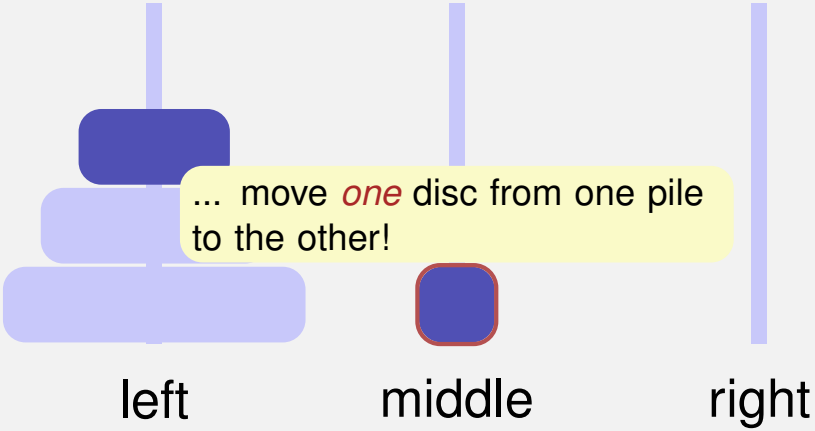
The Towers of Hanoi – Recursive Approach



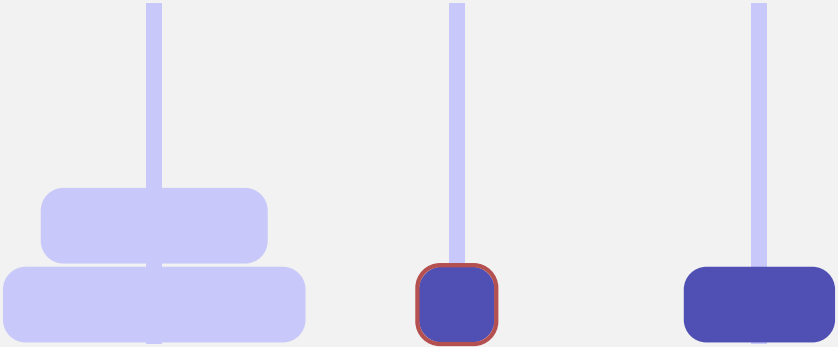
The Towers of Hanoi – Recursive Approach



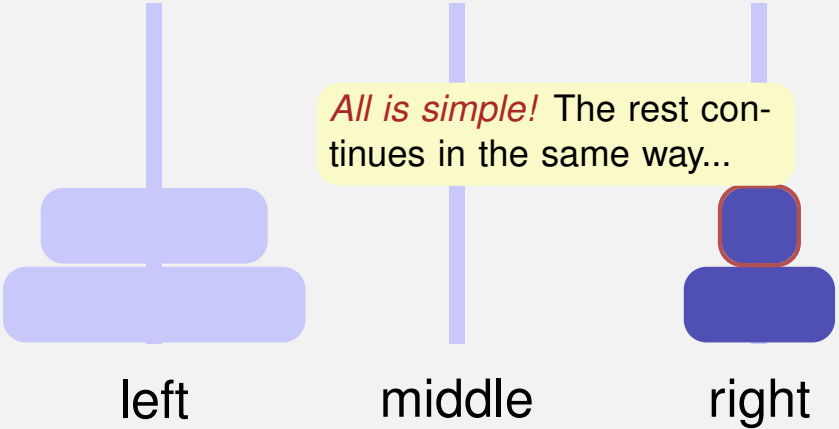
The Towers of Hanoi – Recursive Approach



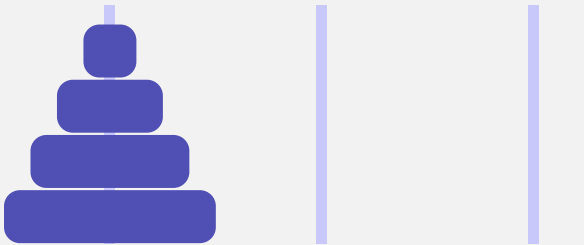
The Towers of Hanoi – Recursive Approach



The Towers of Hanoi – Recursive Approach



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?

5. Structs

Structs

Structs

A `struct` is a bundle of stuff

Structs

A `struct` is a bundle of stuff

- That could be variables, functions, other structs, and much more ("members")
- The types do not have to be the same
- Offer us a way to define new "objects", e.g. your own number type or mathematical objects such as lines, squares, circles, etc.
- Important: Do not forget the `;` at the end of the definition

Structure of `struct`

```cpp
struct Person {
    unsigned int age;
    std::string field;
    std::vector<int> lucky_nums;
};

int main () {
    Person Adel   = {26, "Computer Science", {42, 161}};
    Person Deli   = Adel;
    Person Jules  = {25, "Linguistics", {13, 12}};
    Person Lily   = {19, "Computational Science", {9, 19}};
    std::cout << "Adel's " << Adel.age <<
              " years old\n" << std::endl;
    return 0;
}
```

Questions?

```
// Optional Subtask 6: overloading the << operators-----
std::ostream& operator<<(std::ostream& os, const vec& v) {
  os << "(" << v.x << "," << v.y << "," << v.z << ")";
  return os;
}
std::ostream& operator<<(std::ostream& os, const line& l) {
  os << l.start << " <-> " << l.end;
  return os;
}
//-------------------------------------------------------
```

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
- Implement a solution (optionally in groups)

Solution to "Geometry Exercise"

Solution to "Geometry Exercise"

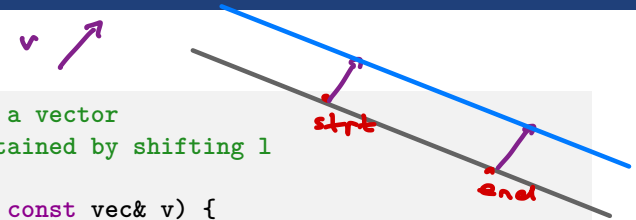
```
// Subtask 1: adding vectors
// POST: returns the sum of a and b
vec sum(const vec& a, const vec& b) {
  // version 1: compact, used for the rest of the example
  return {a.x + b.x, a.y + b.y, a.z + b.z};

  // version 2: longer but maybe easier to understand
  // vec tmp;
  // tmp.x = a.x + b.x;
  // tmp.y = a.y + b.y;
  // tmp.z = a.z + b.z;
  // return tmp;
}
```

Solution to "Geometry Exercise"

```
// Subtask 2: defining a line in 3D
struct line {
  vec start;
  vec end;  // INV: start != end
};
// helper function to print a vector
void print_line(const line& l) {
  print_vec(l.start);
  std::cout << " <-> ";
  print_vec(l.end);
}
```

Solution to "Geometry Exercise"




```
// Subtask 3: shifting line by a vector
// POST: returns a new line obtained by shifting l
// by v.
line shift_line(const line& l, const vec& v) {
  return {sum(l.start, v), sum(l.end, v)};
}

// Subtask 4: overloading the + operator for vectors
vec operator+(const vec& a, const vec& b) {
    return sum(a, b);
}
```

Solution to "Geometry Exercise"

```
// Subtask 5: overloading the + operator for lines
// version 1: use the shift_line function
line operator+(const line& l, const vec& v) {
    return shift_line(l, v);
}

// version 2: make use of the overloaded + operator for vectors
line operator+(const line& l, const vec& v) {
    return {l.start + v, l.end + v};
}
```

Questions?

6. Outro

General Questions?

Till next time!

Cheers!