Move Semantics

We will talk about properties of the move semantic in this lesson that are not mentioned as often.

WE'LL COVER THE FOLLOWING

- std::move
- STL
  - Example
- User-defined data types
  - Example
- Strategy of the move constructor
  - The move constructor and the other big five

Containers of the STL can have non-copyable elements. The copy semantic is the fallback for the move semantic.

Let's learn more about the move semantic.

std::move

The function `std::move` moves its resource.

- Needs the header `<utility>`.
- Converts the type of its argument into a rvalue reference.
- The compiler applies the move semantic to the rvalue reference.
- Is a `static_cast` to a rvalue reference under the hood.

```
static_cast<std::remove_reference<decltype(arg)>::type&&>(arg);
```

- What is happening here?
  - `decltype(arg)`: deduces the type of the argument.
  - `std::remove_reference<....>` removes all references from the type of

the argument.

- `static_cast<....>&&` adds two references to the type.

> 💡 **Copy semantic is a fallback for move semantic.** This means if we invoke `std::move` with a non-moveable type, copy-semantic is used because an rvalue can be bound to an rvalue reference and a constant lvalue reference.

STL

Each container of the STL and `std::string` gets two new methods:

- Move constructor
- Move assignment operator

These new methods get their arguments as **non-constant** rvalue references.

Example

```
vector{
  vector(vector&& vec);                  //move constructor
  vector& operator = (vector&& vec);     //move assignment
  vector(const vector& vec);             //copy constructor
  vector& operator = (const vector& vec); // copy assignment
}
```

> The classical copy constructor and copy assignment operator get their argument as a **constant** lvalue reference.

User-defined data types

User-defined data types can support move and copy semantics as well.

Example

```
class MyData{
  MyData(MyData&& m) = default;
  MyData& operator = (MyData&& m) = default;
  MyData(const MyData& m) = default;
  MyData& operator = (const myData& m) = default;
};
```

> The move semantic has priority over the copy semantic.

Strategy of the move constructor

1. Set the attributes of the new object.
2. Move the content of the old object.
3. Set the old object in a valid state.

The move constructor and the other big five

- The move constructor is created automatically if all attributes of the class and all base classes also have one move constructor.
- This rule holds for the big six:
  - Default constructor
  - Destructor
  - Move and copy constructor
  - Move and copy assignment operator

Let's see a few examples in the next lesson.