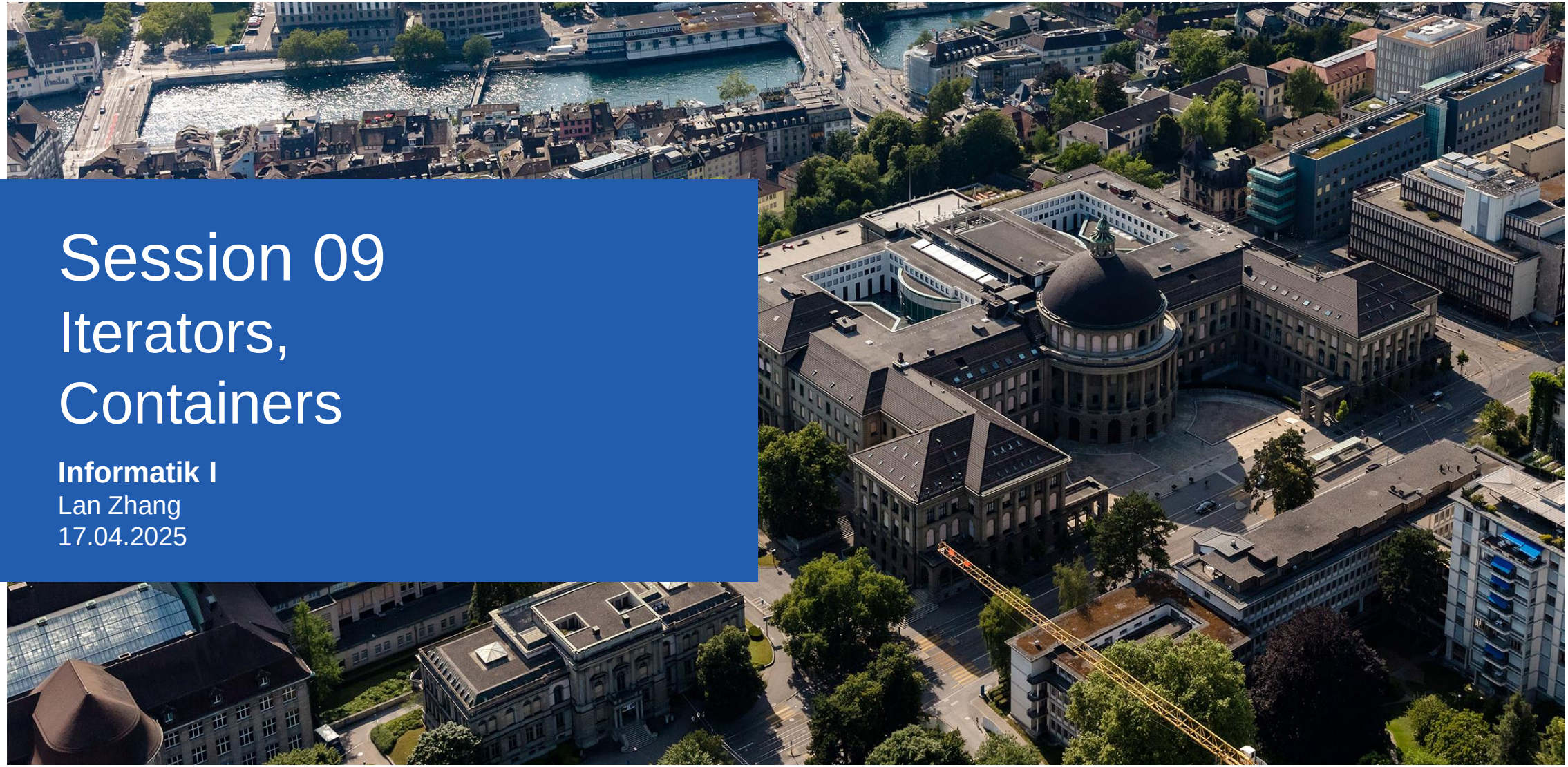
ETH zürich



Session 09
Iterators, Containers

Informatik I

Lan Zhang

17.04.2025

Schedule

1. Exercise Feedback
2. Theory Recap
   - Structs & Classes
   - Containers
   - Iterators
3. In-Class Code Examples
   - Tribool
   - The algorithm Library
   - Optional: Recursion to Iteration

Structs & Classes

Member functions

- provide functionality for a class
- access to private class members
- declaration inside class
- definition also possible externally
  - enables precompiled libraries
  - affiliation to the class must be explicitly mentioned by means of the :: notation

Constructors

- Special member functions that create (“instantiate”) objects
- Has the same name as the class and no return type
- Can be overloaded (defined in multiple ways)

```
struct Complex{
   double real;
   double imag;
   // Default Constructor
   Complex(){
        real = 0;
        imag = 1;
   }
 };
Complex z; // identical: Complex z = Complex();
```

Constructors

- Special member functions that create (“instantiate”) objects
- Has the same name as the class and no return type
- Can be overloaded (defined in multiple ways)

```
struct Complex{
    // General Constructor
    Complex (double x, double y){
        real = x;
        imag = y;
    }
    // Equivalent to Initializer List
    Complex(double x, double y): real(x), imag(y){
    }
  };
  Complex z = Complex(0,1); // z = i
```

Constructors

- Special member functions that create (“instantiate”) objects
- Has the same name as the class and no return type
- Can be overloaded (defined in multiple ways)

```
struct Complex{
  public:
     // Conversion Constructor (double -> Complex)
     Complex(const double i) : real (i), imag (0)
  private:
   double real;
   double imag;
  };
```

In-Class Code Example

Tribool: 3-valued logic (false F, unknown U, true T)

1. (int) Constructor
2. (std::string) Constructor
3. Implement string() method
4. Implement the logical AND operator

Reminder:
Declarations into Tribool.h
Definitions into Tribool.cpp

Why is it a good idea to have value as private member?

- ensure it is always 0, 1, or 2 (invalid write-access)
- change type from 32bit int to char without affecting the public interface of the class

In-Class Code Example

Tribool: 3-valued logic (false F, unknown U, true T)

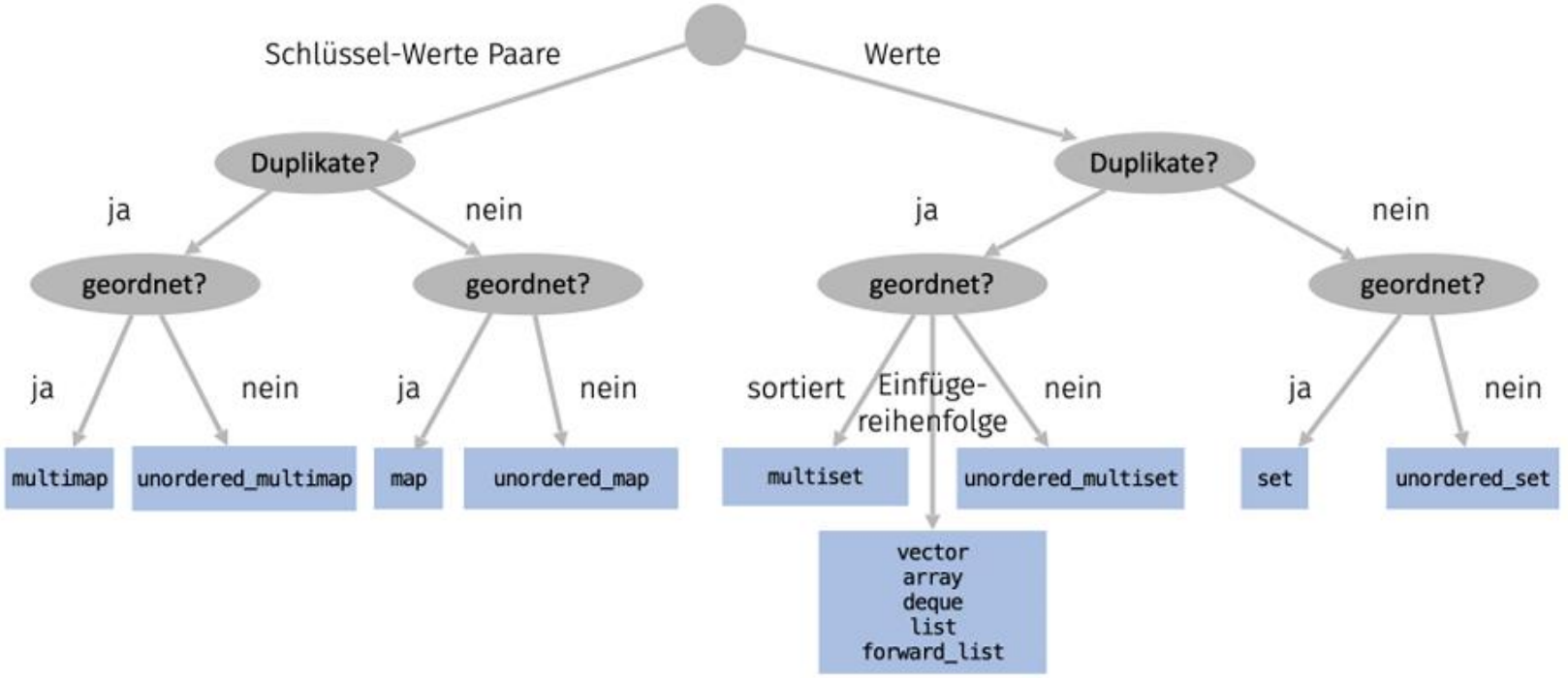
When do we use const member functions?

The const refers to *this. It promises that the value of the implicit argument is not changed by the function execution.

```
class Insurance {
  public:
     double get_value() const {
            return value; // same: return (*this).value;
     }
     ... // e.g. members which set the data members
   private:
     double value;
};
```

Containers

- Objects used to store collections of elements
- Default operations on the collection



[Containers library - cppreference.com]

Containers: Set

Data type for sets (each element occurs only once)

```
#include<set>
#include<unordered_set>
std::set<T> = {A, B, C}; // ordered set where elements are stored in a specific order
std::unordered_set<T> = {B, C, A};
std::set<int> set (b, e); // set with values in range [b,e) given by iterators of another container
```

Set iterators work like vector iterators, but certain operations are undefined:

```
[], +, -, <, >, <=, >=, +=, -=
```

i.e. no random access or arithmetic operations

```
To move only: ++..., ...++, --..., ...--, =
To compare only: ==, !=
```

Containers: Set

Commonly used functions

`set.clear()`	-> empty/reset a set
`set.empty()`	-> check if set is empty
`set.find(e)`	-> returns iterator to found element or set.end() if element is not contained
`set.insert(e)`	-> add element to set
`set1.insert(set2.begin(), set2.end())`	-> insert range of values from another set

Iterators

- Used to iterate over or access elements in a container in a unified way
- Does not require knowledge of internal implementation details of a container
- Container-type specific (e.g. T::iterator)

Given a container c:

- `it = c.begin()`: Iterator pointing to the first element
- `it = c.end()`: Iterator pointing behind the last element
- `*it`: Access target element (by dereference of the iterator)
- `++it`: Advance iterator by one element

const (Iterator)

`T::const_iterator ...` // forbids write access to the object i.e. *itr = … invalid

`const T::iterator …` // forbids modification of the iterator‘s position i.e. itr += … invalid

Iterators: Traversal

```
std::vector<int> v(3);
// with random access
for (int i = 0; i < v.size(); i++){
      std::cout<< v.at(i) << std::endl;
}
// with iterators
for (auto it = v.begin(); it != v.end(); it++){
      std::cout << *it << std::endl;
}
// with iterator-based loop
 for(int& i : v) i += 3;
 for(int i : v) std::cout << i; // 333
```

In-Class Code Example

The algorithm Library

```
#include <algorithm>
std::fill(b,p,val) // Read a value into an interval [b,p)
```

`std::find(b,p, val)` // An iterator is returned on the first occurrence found. If val is not found, it returns the past-the-end iterator p. (Note: past-the-end refers to the range [b,p)).

`std::sort(b,e)` // Sort range [b,e) (only works on containers with random access iterators)

```
std::min_element(b, p) // Iterator to the (first) minimum in range [b,p)
std::max_element(b, p) // Iterator to the (first) maximum in range [b,p)
```

In-Class Code Example

```
// Prints all items in a string that are separated by a common delimiter.
void parse(string parseString, string delimiter) {
  string value;
  int startPos = 0, pos = parseString.find(delimiter);
  while (pos != string::npos){
    value = parseString.substr(startPos, pos - startPos);
    cout << value << endl;
    startPos = pos + delimiter.length();
    pos = parseString.find(delimiter, startPos);
  }
  value = parseString.substr(startPos, parseString.length() - startPos);
  cout << value << endl;
}
```

Example call: parse("this::is::a::test", "::");