Informatik - Exercise Session
Classes and Iterators

Classes vs. structs

What is the difference between a `class` und `struct`?

Classes vs. structs

What is the difference between a `class` und `struct`?

The *only* difference is the default visibility of members: A `struct` has default visibility `public`, a `class` has default visibility `private`:

Classes vs. structs

What is the difference between a `class` und `struct`?

The *only* difference is the default visibility of members: A `struct` has default visibility `public`, a `class` has default visibility `private`:

```
class PrivateAccess {
    int a;
};
```

is the same as

```
struct PrivateAccess {
    private:
        int a;
};
```

Classes vs. structs

What is the difference between a `class` und `struct`?

The *only* difference is the default visibility of members: A `struct` has default visibility `public`, a `class` has default visibility `private`:

```
class PrivateAccess {
    int a;
};
```

is the same as

```
struct PrivateAccess {
    private:
        int a;
};
```

Or the other way around:

```
class PublicAccess {
    public:
        int a;
};
```

is the same as

```
struct PublicAccess {
    int a;
};
```

When to use . vs ::

The double colon `::` operator was exclusively used to access members of other `namespaces` until now, like in `std::cout`.

When to use . vs ::

The double colon `::` operator was exclusively used to access members of other `namespaces` until now, like in `std::cout`.

It is also used to access members
of `classes` or `structs`:

```
class Foo {
    void bar();
}

void Foo::bar() {
    // ...
}
```

When to use . vs ::

The double colon `::` operator was exclusively used to access members of other `namespaces` until now, like in `std::cout`.

It is also used to access members of `classes` or `structs`:

```
class Foo {
    void bar();
}

void Foo::bar() {
    // ...
}
```

This is not the same as the dot `.` operator, which is used to access members of *objects* (instances of `classes`):

```
int main() {
    Foo f = Foo();
    f.bar();
}
```

Iterators

`https://en.cppreference.com/w/cpp/container` lists all containers in the C++ standard library (e.g. list, set, unordered set, . . .).

Iterators

`https://en.cppreference.com/w/cpp/container` lists all containers in the C++ standard library (e.g. list, set, unordered set, . . .).

`https://en.cppreference.com/w/cpp/algorithm` lists all algorithms in the C++ standard library (e.g. max, max element, sort, . . .).

Iterators

`https://en.cppreference.com/w/cpp/container` lists all containers in the C++ standard library (e.g. list, set, unordered set, . . .).

`https://en.cppreference.com/w/cpp/algorithm` lists all algorithms in the C++ standard library (e.g. max, max element, sort, . . .).

Iterators are used to move through ($\rightarrow$ "iterate" over) elements in a container without knowing how the data is stored. Usage:

Iterators

`https://en.cppreference.com/w/cpp/container` lists all containers in the C++ standard library (e.g. list, set, unordered set, . . .).

`https://en.cppreference.com/w/cpp/algorithm` lists all algorithms in the C++ standard library (e.g. max, max element, sort, . . .).

Iterators are used to move through ($\rightarrow$ “iterate” over) elements in a container without knowing how the data is stored. Usage:

- `it = c.begin()`

Iterators

`https://en.cppreference.com/w/cpp/container` lists all containers in the C++ standard library (e.g. list, set, unordered set, . . .).

`https://en.cppreference.com/w/cpp/algorithm` lists all algorithms in the C++ standard library (e.g. max, max element, sort, . . .).

Iterators are used to move through ($\rightarrow$ “iterate” over) elements in a container without knowing how the data is stored. Usage:

- `it = c.begin()` points to the first element.

Iterators

`https://en.cppreference.com/w/cpp/container` lists all containers in the C++ standard library (e.g. list, set, unordered set, . . .).

`https://en.cppreference.com/w/cpp/algorithm` lists all algorithms in the C++ standard library (e.g. max, max element, sort, . . .).

Iterators are used to move through ($\rightarrow$ "iterate" over) elements in a container without knowing how the data is stored. Usage:

- `it = c.begin()` points to the first element.
- `it = c.end()`

Iterators

`https://en.cppreference.com/w/cpp/container` lists all containers in the C++ standard library (e.g. list, set, unordered set, ...).

`https://en.cppreference.com/w/cpp/algorithm` lists all algorithms in the C++ standard library (e.g. max, max element, sort, ...).

Iterators are used to move through ($\rightarrow$ "iterate" over) elements in a container without knowing how the data is stored. Usage:

- `it = c.begin()` points to the first element.
- `it = c.end()` points *behind* the last element.

Iterators

`https://en.cppreference.com/w/cpp/container` lists all containers in the C++ standard library (e.g. list, set, unordered set, . . .).

`https://en.cppreference.com/w/cpp/algorithm` lists all algorithms in the C++ standard library (e.g. max, max element, sort, . . .).

Iterators are used to move through ($\to$ "iterate" over) elements in a container without knowing how the data is stored. Usage:

- `it = c.begin()` points to the first element.
- `it = c.end()` points *behind* the last element.
- `*it`

Iterators

`https://en.cppreference.com/w/cpp/container` lists all containers in the C++ standard library (e.g. list, set, unordered set, . . .).

`https://en.cppreference.com/w/cpp/algorithm` lists all algorithms in the C++ standard library (e.g. max, max element, sort, . . .).

Iterators are used to move through ($\rightarrow$ "iterate" over) elements in a container without knowing how the data is stored. Usage:

- `it = c.begin()` points to the first element.
- `it = c.end()` points *behind* the last element.
- `*it` accesses the element where the iterator currently points.

Iterators

`https://en.cppreference.com/w/cpp/container` lists all containers in the C++ standard library (e.g. list, set, unordered set, . . .).

`https://en.cppreference.com/w/cpp/algorithm` lists all algorithms in the C++ standard library (e.g. max, max element, sort, . . .).

Iterators are used to move through ($\rightarrow$ "iterate" over) elements in a container without knowing how the data is stored. Usage:

- `it = c.begin()` points to the first element.
- `it = c.end()` points *behind* the last element.
- `*it` accesses the element where the iterator currently points.
- `++it`

Iterators

`https://en.cppreference.com/w/cpp/container` lists all containers in the C++ standard library (e.g. list, set, unordered set, . . .).

`https://en.cppreference.com/w/cpp/algorithm` lists all algorithms in the C++ standard library (e.g. max, max element, sort, . . .).

Iterators are used to move through ($\rightarrow$ "iterate" over) elements in a container without knowing how the data is stored. Usage:

- `it = c.begin()` points to the first element.
- `it = c.end()` points *behind* the last element.
- `*it` accesses the element where the iterator currently points.
- `++it` advances the iterator by one element.