Informatik - Exercise Session
An overview on some syntactic sugars

Calling constructors

Assuming we have a `class Test` with a constructor taking one integer argument, the following are equivalent:

```
1 Test t1 = Test(1);
2 Test t2 = Test{1};
3 Test t3 = {1};
4 Test t4{1};
```

Calling constructors

Assuming we have a `class Test` with a constructor taking one integer argument, the following are equivalent:

```
1 Test t1 = Test(1);
2 Test t2 = Test{1};
3 Test t3 = {1};
4 Test t4{1};
```

If we have an additional (default) constructor without any arguments, there is a fifth way to initialize an object, beside removing the integer from all the ones mentioned above:

```
1 Test t5;
```

Constructor syntax

When writing a constructor, we often want to directly initialize some member variables with arguments given to the constructor:

```
1  class Foo {
2    int bar;
3    int baz;
4    public:
5      Foo(int i, int j) {
6        this->bar = i;
7        this->baz = j;
8      }
9  };
```

Constructor syntax

When writing a constructor, we often want to directly initialize some member variables with arguments given to the constructor:

```
1  class Foo {
2    int bar;
3    int baz;
4    public:
5      Foo(int i, int j) {
6        this->bar = i;
7        this->baz = j;
8      }
9  };
```

```
1  class Foo {
2    int bar;
3    int baz;
4    public:
5      Foo(int i, int j) : bar(i), baz(j) {}
6  };
```

We can shorten this example using the `:` notation for constructors.

Overloaded operators

Depending on where exactly an operator was overloaded, we can call it using the explicit function syntax instead of the infix version.

Overloaded operators

Depending on where exactly an operator was overloaded, we can call it using the explicit function syntax instead of the infix version.

For an overloaded operator outside of any class:

`Test operator+(const Test& a, const Test& b);`, these are equivalent:

```
// Test t1, t2;
t1 + t2;            // infix
operator+(t1, t2); // function
```

Overloaded operators

Depending on where exactly an operator was overloaded, we can call it using the explicit function syntax instead of the infix version.

For an overloaded operator outside of any class:

`Test operator+(const Test& a, const Test& b);`, these are equivalent:

```
// Test t1, t2;
t1 + t2;            // infix
operator+(t1, t2); // function
```

For an overloaded operator as a method:

`Test Test::operator+(const Test& other) const;`, these are equivalent:

```
// Test t1, t2;
t1 + t2;           // infix
t1.operator+(t2); // function
```

Overloaded operators

Depending on where exactly an operator was overloaded, we can call it using the explicit function syntax instead of the infix version.

For an overloaded operator outside of any class:

`Test operator+(const Test& a, const Test& b);`, these are equivalent:

```
// Test t1, t2;
t1 + t2;            // infix
operator+(t1, t2); // function
```

For an overloaded operator as a method:

`Test Test::operator+(const Test& other) const;`, these are equivalent:

```
// Test t1, t2;
t1 + t2;           // infix
t1.operator+(t2); // function
```

Whenever possible, use the infix notation, where you don't have to know where and how the operator was overloaded.