



Grundlagen der Informatik und Programmierung 2

Templates

Parameter und ihre Typen

Prof. Dr. Tom Vierjahn

Visual Computing (<https://vc.w-hs.de>)
Fachbereich Wirtschaft und Informationstechnik
Campus Bocholt

Sommersemester 2020



Template-Funktion less_than:

```
template <typename T>
bool less_than(T a, T b) {
    return a < b;
}
```

Aufruf

```
std::cout << std::boolalpha;
std::cout << less_than(5, 4) << std::endl;
std::cout << less_than(3.1415926, 4.123) << std::endl;
```

Ausgabe

Template-Funktion less_than:

```
template <typename T>
bool less_than(T a, T b) {
    return a < b;
}
```

Aufruf

```
const char* a = "Meyer";
const char* b = "Meier";
std::cout << std::boolalpha;
std::cout << less_than(std::string(a), std::string(b)) << std::endl;
std::cout << less_than(a, b) << std::endl;
```

Ausgabe

Template-Funktion less_than:

```
template <typename T>
bool less_than(T a, T b) {
    return a < b;
}
```

Aufruf

```
const char* c = "Meier";
const char* d = "Meyer";
std::cout << std::boolalpha;
std::cout << less_than(std::string(c), std::string(d)) << std::endl;
std::cout << less_than(c, d) << std::endl;
```

Ausgabe

Template-Funktion less_than mit Spezialisierung für const char*:

```
template <typename T>
bool less_than(T a, T b) {
    return a < b;
}
template <>
bool less_than<const char*>(const char* a, const char* b) {
    return strcmp(a, b) < 0;
}
```

- Spezialisierung muss nach Erzeugung des Templates erfolgen.

Aufruf

```
const char* a = "Meyer";  
const char* b = "Meier";  
std::cout << std::boolalpha;  
std::cout << less_than(std::string(a), std::string(b)) << std::endl;  
std::cout << less_than(a, b) << std::endl;
```

Ausgabe

Aufruf

```
const char* c = "Meier";  
const char* d = "Meyer";  
std::cout << std::boolalpha;  
std::cout << less_than(std::string(c), std::string(d)) << std::endl;  
std::cout << less_than(c, d) << std::endl;
```

Ausgabe

Was ist hiermit gemeint?

```
less_than(true, false);
```

- ▶ Aussagekraft dieses Vergleichs zweifelhaft.
- ▶ Instanziierung lässt sich verhindern.

Template mit static_assert:

```
template <typename T>
bool less_than(T a, T b) {
    static_assert(!std::is_same<bool, T>::value,
                  "Comparing bools using less_than<bool> is not sensible.");
    return a < b;
}
```

- ▶ Compiler bricht mit Fehlermeldung ab.

Beispiel aus der Standardbibliothek: `std::map`:

```
std::map<unsigned int, std::string> students;  
  
students[199100001] = "Jane Appleseed";  
students[199100002] = "John Doe";  
students[199100003] = "J. Random Student";  
  
std::cout << students[199100002] << std::endl;
```

Ausgabe:

- ▶ Template-Parameter können Werte aufnehmen.
- ▶ Zulässig sind u. a. Ganzzahlen, Pointer, Pointer auf Member, enums.

Beispiel:

```
template <typename T, unsigned int dimensions>
struct Vec {
    T data[dimensions];
};
```

Anwendung:

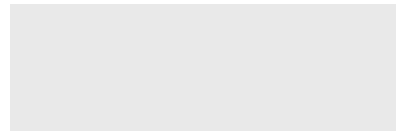
operator<<:

```
template <typename T, unsigned int dimensions>
std::ostream& operator<<(std::ostream& os, const Vec<T, dimensions>& v) {
    os << '[';
    for (unsigned int i = 0; i < dimensions; ++i) {
        os << v.data[i];
        os << (i == (dimensions - 1) ? ']' : '\t');
    }
    return os;
}
```

Anwendung:

```
std::cout << vf1 << std::endl;
std::cout << vf2 << std::endl;
std::cout << vf3 << std::endl;
```

Ausgabe:



- ▶ Spezialisierungen
- ▶ Compile-Zeit-Assertions (`static_assert`)
- ▶ mehrere Template-Parameter
- ▶ Werte als Template-Parameter

Prof. Dr. Tom Vierjahn

► E-Mail: tom.vierjahn@w-hs.de

Visual Computing

► Web: <https://vc.w-hs.de>

► YouTube: Visual Computing WH

► Twitter: @VisComputingWH

Westfälische Hochschule

Fachbereich Wirtschaft und Informationstechnik

Campus Bocholt



Veröffentlicht unter der Creative-Commons-Lizenz

Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)