

# Algorithmen und Datenstrukturen in C/C++

Prof. Dr. rer. nat. Peter Gerwinski

7. Mai 2020

# Algorithmen und Datenstrukturen in C/C++

<https://gitlab.cvh-server.de/pgerwinski/ad.git>

## 1 Einführung

## 2 Einführung in C++

...

### 2.4 Referenz-Typen

### 2.5 Überladbare Operatoren und Funktionen

### 2.6 Namensräume

### 2.7 Objekte

### 2.8 Strings

### 2.9 Templates

### 2.10 Exceptions

## 3 Datenorganisation

- Listen, Bäume, Hash-Tabellen, ...

## 4 Datenkodierung

- Fehlererkennung und -korrektur
- Kompression
- Kryptographie

## 5 Hardwarenahe Algorithmen

## 6 Optimierung

## 7 Numerik



Änderungen  
vorbehalten

## 2.8 Strings

- **#include** <string>
- String-Klasse
- String-Konstante sind **const char \***
- C-kompatiblen String extrahieren: `c_str ()`
- In String schreiben: **#include** <sstream>, `ostringstream`

## 2.9 Templates

Anwendung desselben Quelltextes auf verschiedene Datentypen

- `template <typename x> ...`
- `template <class x> ...`

## 2.9 Templates

Anwendung desselben Quelltextes auf verschiedene Datentypen

- `template <typename x> ...`
- `template <class x> ...`

Vorsicht: Fehler werden erst bei Instantiierung erkannt!

## 2.9 Templates

Anwendung desselben Quelltextes auf verschiedene Datentypen

- `template <typename x> ...`
- `template <class x> ...`

Vorsicht: Fehler werden erst bei Instantiierung erkannt!

- Template-Spezialisierung:  
`template <> foo <int> ...`

## 2.10 Exceptions

```
try
{
    ...
    throw <value>;
    ...
}
catch (<type> <variable>)
{
    ...
}
catch ...
```

- Nach den `catch()`-Statements wird, soweit nicht anders programmiert, das Programm fortgesetzt.
- `throw;` (ohne Wert):  
an übergeordneten Exception-Handler weiterreichen
- C-Äquivalent:  
`setjmp()`, `longjmp()`
- speziell für `<type>`:  
Nachfahren von `class exception`
- veraltet:  
*dynamic exception specifications*