

# Algorithmen und Datenstrukturen in C/C++

Prof. Dr. rer. nat. Peter Gerwinski

23. April 2018

# Algorithmen und Datenstrukturen in C/C++

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

## 1 Einführung

## 2 Einführung in C++

## 3 Datenorganisation

- Listen, Bäume, Hash-Tabellen, ...

## 4 Datenkodierung

- Fehlererkennung und -korrektur
- Kompression
- Kryptographie

## 5 Hardwarenahe Algorithmen

- FFT, CORDIC, ...

## 6 Optimierung

- Wegfindung, ...

## 7 Numerik



Änderungen  
vorbehalten

# Algorithmen und Datenstrukturen in C/C++

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

## 1 Einführung

## 2 Einführung in C++

...

### 2.3 Referenz-Typen

### 2.4 Überladbare Operatoren und Funktionen

### 2.5 Namensräume

### 2.6 Objekte

### 2.7 Strings

### 2.8 Templates

### 2.9 Exceptions

## 3 Datenorganisation

## 4 Datenkodierung

## 5 Hardwarenahe Algorithmen

## 6 Optimierung

## 7 Numerik



Änderungen  
vorbehalten

## 2.5 Namensräume

```
#include <iostream>
```

```
using namespace std;
```

```
int main ()  
{  
    cout << "Hello,_world!" << endl;  
    return 0;  
}
```

```
namespace my_output  
{  
    ...  
}
```

```
using namespace my_output;
```

## 2.6 Objekte

```
struct TBase  
{  
};
```

```
struct TInteger: public TBase  
{  
    int content;  
};
```

```
struct TString: public TBase  
{  
    char *content;  
};
```

## 2.6 Objekte: Zugriffsrechte

- `public`, `private`, `protected`  
nicht nur Bürokratie, sondern auch Kapselung  
(Maßnahme gegen „Namensraumverschmutzung“)
- **`struct`**: standardmäßig `public`  
`class`: standardmäßig `private`
- `friend`-Funktionen und -Klassen
- Klasse als Namensraum:  
**`static`**-„Member“-Variable  
**`static`**-„Methoden“  
Deklarationen von z. B. Konstanten und Typen

## 2.6 Objekte: Konstruktoren und Destruktoren

- leerer Standard-Konstruktor
- *Copy-Konstruktor*
- Konstruktor-Aufruf als „Initialisierung“
- Konstruktor-Aufruf mit `new`  
Destruktor-Aufruf mit `delete`
- automatischer Destruktor-Aufruf  
beim Verlassen des Gültigkeitsbereichs

## 2.7 Strings

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



## 2.8 Templates

Anwendung desselben Quelltextes auf verschiedene Datentypen

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

## 2.8 Templates

Anwendung desselben Quelltextes auf verschiedene Datentypen

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

Vorsicht: Fehler werden erst bei Instantiierung erkannt!

## 2.8 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> ...`

## 2.9 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*

# Algorithmen und Datenstrukturen in C/C++

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

## 1 Einführung

## 2 Einführung in C++

...

### 2.5 Namensräume

### 2.6 Objekte

### 2.7 Strings

### 2.8 Templates

### 2.9 Exceptions

## 3 Datenorganisation

- Listen, Bäume, Hash-Tabellen, ...

## 4 Datenkodierung

## 5 Hardwarenahe Algorithmen

## 6 Optimierung

## 7 Numerik



Änderungen  
vorbehalten