

Algorithmen und Datenstrukturen in C/C++

Prof. Dr. rer. nat. Peter Gerwinski

26. April 2018

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

2.10 Typ-Konversionen

3 Datenorganisation

3.1 Standard-Container-Templates

3.2 Iteratoren

...

4 Datenkodierung

5 Hardwarenahe Algorithmen

6 Optimierung

7 Numerik



Änderungen
vorbehalten

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> ...`

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

2.10 Typ-Konversionen

- In C:

```
char *hello = "Hello, world!";  
uint64_t address = (uint64_t) hello;  
printf ("%s" PRIu64 "\n", address);
```

- alternative Syntax in C++:

```
char *hello = "Hello, world!";  
uint64_t address = uint64_t (hello);  
cout << address << endl;
```

- zusätzlich in C++:

implizite und explizite Typumwandlung zwischen Zeigern auf Klassen

```
dynamic_cast<>()  
static_cast<>()  
reinterpret_cast<>()  
const_cast<>()
```

2.10 Typ-Konversionen

<http://www.cplusplus.com/doc/tutorial/typecasting/>

- Zuweisung: Zeiger auf abgeleitete Klasse an Zeiger auf Basisklasse
→ implizite Typumwandlung möglich
- Zuweisung: Zeiger auf Basisklasse an Zeiger auf abgeleitete Klasse
→ nur explizite Typumwandlung möglich:
`dynamic_cast<>()`, `static_cast<>()`
- implizite Typumwandlungen in der Klasse definieren:
 - Initialisierung durch Konstruktor
 - Zuweisungs-Operator
 - Typumwandlungsoperator
- implizite Typumwandlungen ausschalten:
Schlüsselwort `explicit`

2.10 Typ-Konversionen

<http://www.cplusplus.com/doc/tutorial/typecasting/>

- `dynamic_cast<>()`
Zuweisung: Zeiger auf Basisklasse an Zeiger auf abgeleitete Klasse
explizite Typumwandlung mit Prüfung, ggf. Exception
- `static_cast<>()`
Zuweisung: Zeiger auf Basisklasse an Zeiger auf abgeleitete Klasse
explizite Typumwandlung ohne Prüfung
- `reinterpret_cast<>()`
Typumwandlung ohne Prüfung zwischen Zeigern untereinander
und zwischen Zeigern und Integer-Typen
- `const_cast<>()`
„**const**“ ein- bzw. ausschalten

3 Datenorganisation

3.1 Standard-Container-Templates

<http://www.cplusplus.com/reference/stl/>

- Fertige Templates für Arrays (Stack, FIFO), verkettete Listen – und mehr
- Methoden:
 `push_back ()`, `pop_back ()`,
 `push_front ()`, `pop_front ()`,
 `insert ()`, `delete ()`, ...
- `emplace`-Methoden
- Der Container übernimmt die vollständige Kontrolle über seinen Inhalt.

3.1 Standard-Container-Templates

array	Array mit fester Größe
bitset	festes Array von Bits (Booleans)
vector	dynamisches Array
vector <bool>	dynamisches Bit-Array
forward_list	einfach-verkettete Liste
list	doppelt-verkettete Liste
set	binärer Baum
multiset	mehrfache Elemente zulässig
unordered_set	Hash-Tabelle
unordered_multiset	mehrfache Elemente zulässig
map	binärer Baum mit separaten Schlüsselwerten
multimap	mehrere Elemente pro Schlüssel
unordered_map	Hash-Tabelle mit separaten Schlüsselwerten
unordered_multimap	mehrere Elemente pro Schlüssel
stack	Stack
queue	FIFO
deque	<i>double-ended queue</i>
priority_queue	geordneter Push-Pop-Container

3 Datenorganisation

3.2 Iteratoren

Pointer-Arithmetik:

```
int prime[5] = { 2, 3, 5, 7, 11 };  
for (int *p = prime; p != prime + 5; p++)  
    cout << *p << endl;
```

Iterator als Verallgemeinerung:

```
array<int, 5> prime = { { 2, 3, 5, 7, 11 } };  
for (array<int, 5>::iterator p = prime.begin (); p != prime.end (); p++)  
    cout << *p << endl;
```

3 Datenorganisation

3.3 Hash-Tabellen

Idee: Ein String ist auch nur eine große Zahl.
Wenn man die irgendwie zurechtstutzt,
kann man sie als Array-Index verwenden.

Übungsaufgabe:

Schreiben Sie ein Programm, das zu Strings (z. B. Namen)
Datensätze speichern und diese in $\mathcal{O}(1)$ wieder abrufen kann.

- (a) Verwenden Sie das Hash-Tabellen-Template
aus der C++-Standardbibliothek.
- (b) Programmieren Sie die Hash-Tabelle selbst.

Teamarbeit, Internet-Recherchen
und die Verwendung anderer Container-Standard-Templates
sind ausdrücklich erlaubt.

Algorithmen und Datenstrukturen in C/C++

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

1 Einführung

2 Einführung in C++

...

2.7 Strings

2.8 Templates

2.9 Exceptions

2.10 Typ-Konversionen

3 Datenorganisation

3.1 Standard-Container-Templates

3.2 Iteratoren

3.3 Hash-Tabellen

3.4 Balancierte Bäume

...

4 Datenkodierung

5 Hardwarenahe Algorithmen

6 Optimierung

7 Numerik



Änderungen
vorbehalten