

Algorithmen und Datenstrukturen in C/C++

Prof. Dr. rer. nat. Peter Gerwinski

20. Juni 2024

Algorithmen und Datenstrukturen in C/C++

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

- 1 Einführung
- 2 Arrays und Zeiger für Fortgeschrittene
- 3 Langzahl-Arithmetik
- 4 Kryptographie
- 5 Datenkompression
- 6 Einführung in C++
- 7 Datenorganisation

...



Änderungen
vorbehalten

5 Datenkompression

5.2 Algorithmen

Verlustfreie Kompression

- Run-Length Encoding (RLE)
- Entropie-Kodierung
 - Morsecode
 - UTF-8
 - Huffman-Kodierung
 - Lempel-Ziv-Welch-Algorithmus
- Differenzbildung

Verlustbehaftete Kompression

- JPEG

Weitere Aspekte

- Pixel- vs. Vektorgrafik
- E-Mail-Attachment vs. HTTP

Hochschule Bochum
Bochum University
of Applied Sciences



Hochschule Bochum
Bochum University
of Applied Sciences



Hochschule Bochum
Bochum University
of Applied Sciences



6 Einführung in C++

6.0 Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen



Hardware und/oder Betriebssystem

- Hardware direkt ansprechen und effizient einsetzen
- ... bis hin zu komplexen Software-Projekten

→ Man kann Computer vollständig beherrschen.

6 Einführung in C++

6.0 Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen
- Hardware direkt ansprechen und effizient einsetzen
- ... bis hin zu komplexen Software-Projekten
- leistungsfähig, aber gefährlich

„High-Level-Assembler“

C makes it easy to shoot yourself in the foot.

- kein „Fallschirm“
- kompakte Schreibweise

Bjarne Stroustrup, ca. 1986

http://www.stroustrup.com/bs_faq.html
[#really-say-that](#)

Unix-Hintergrund

- Baukastenprinzip
- konsequente Regeln
- kein „Fallschirm“

6 Einführung in C++

6.1 Was ist C++?

Etabliertes Profi-Werkzeug

- kompatibel zu C

C++ is a better C.

C++ unterstützt

- *objektorientierte Programmierung*
- *Datenabstraktion*
- *generische Programmierung*

Bjarne Stroustrup, Autor von C++
<http://www.stroustrup.com/C++.html>

*C makes it easy to shoot yourself in the foot;
C++ makes it harder, but when you do
it blows your whole leg off.*

Bjarne Stroustrup, Autor von C++, ca. 1986
http://www.stroustrup.com/bs_faq.html
[#really-say-that](#)

Motivation:

Vermeidung unsicherer Techniken,
insbesondere von Präprozessor-Konstruktionen und Zeigern,
unter Beibehaltung der Effizienz

6.2 Elementare Neuerungen in C++ gegenüber C

6.2 Elementare Neuerungen in C++ gegenüber C

- Kommentare mit //

6.2 Elementare Neuerungen in C++ gegenüber C

- Kommentare mit //
- Konstante:
const int n = 5;
int prime[n] = { 2, 3, 5, 7, 11 };

6.2 Elementare Neuerungen in C++ gegenüber C

- Kommentare mit //
- Konstante:
`const int n = 5;`
`int prime[n] = { 2, 3, 5, 7, 11 };`
- Ab C++11: `constexpr`-Funktionen
C++11: darf nur aus einem einzigen `return`-Statement bestehen
→ `?:` statt `if`, Rekursion statt Schleife
C++-14: auch Verzweigungen und Schleifen erlaubt

6.2 Elementare Neuerungen in C++ gegenüber C

- Kommentare mit //
- Konstante:
`const int n = 5;`
`int prime[n] = { 2, 3, 5, 7, 11 };`
- Ab C++11: `constexpr`-Funktionen
C++11: darf nur aus einem einzigen `return`-Statement bestehen
→ `?:` statt `if`, Rekursion statt Schleife
C++14: auch Verzweigungen und Schleifen erlaubt
- leere Parameterliste: `void` optional
in C: ohne `void` = Parameterliste wird nicht geprüft

6.2 Elementare Neuerungen in C++ gegenüber C

- Kommentare mit //
- Konstante:
`const int n = 5;`
`int prime[n] = { 2, 3, 5, 7, 11 };`
- Ab C++11: `constexpr`-Funktionen
C++11: darf nur aus einem einzigen `return`-Statement bestehen
→ `?:` statt `if`, Rekursion statt Schleife
C++-14: auch Verzweigungen und Schleifen erlaubt
- leere Parameterliste: `void` optional
in C: ohne `void` = Parameterliste wird nicht geprüft
- Operatoren `new` und `delete`
als Alternative zu den Funktionen `malloc()` und `free()`

6.3 Referenz-Typen

```
void calc_answer (int &answer)
{
    answer = 42;
}
```

... als Alternative zu ...

```
void calc_answer (int *answer)
{
    *answer = 42;
}
```

- Zeiger „verborgen“, übersichtlicher und sicherer
- Es gibt keinen **NULL**-Wert.
→ Für verkettete Listen u. ä.: Tricks erforderlich

6.4 Überladbare Operatoren und Funktionen

```
#include <iostream>
```

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

6.4 Überladbare Operatoren und Funktionen

```
#include <iostream>
```

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

Bemerkungen:

- Compilieren mit `g++` statt `gcc`:
C++-Bibliotheken mit einbinden
- Der Operator `<<` hat normalerweise keinen Seiteneffekt, hier schon.

6.4 Überladbare Operatoren und Funktionen

```
#include <iostream>
```

```
struct vector  
{  
    double x, y, z;  
};
```

```
vector operator + (vector u, vector v)  
{  
    vector w = { u.x + v.x, u.y + v.y, u.z + v.z };  
    return w;  
}
```

- ++ wird zum Präfix-Operator.
- ++ mit zusätzlichem (ungenutzten) **int**-Parameter wird zum Postfix-Operator.

6.4 Überladbare Operatoren und Funktionen

```
void print (const char *s)
```

```
{  
    printf ("%s", s);  
}
```

```
void print (int i)
```

```
{  
    printf ("%d", i);  
}
```

6.4 Überladbare Operatoren und Funktionen

```
void print (const char *s)
```

```
{  
    printf ("%s", s);  
}
```

```
void print (int i)
```

```
{  
    printf ("%d", i);  
}
```

Optionale Parameter:

```
void print (const char *s = "\n")
```

```
{  
    printf ("%s", s);  
}
```

6.4 Überladbare Operatoren und Funktionen

```
void print (const char *s)
```

```
{  
    printf ("%s", s);  
}
```

```
void print (int i)
```

```
{  
    printf ("%d", i);  
}
```

Für den Linker:
veränderte, eindeutige Namen

Optionale Parameter:

```
void print (const char *s = "\n")
```

```
{  
    printf ("%s", s);  
}
```

wird vom Compiler erledigt

6.4 Überladbare Operatoren und Funktionen

```
void print (const char *s)
```

```
{  
    printf ("%s", s);  
}
```

```
void print (int i)
```

```
{  
    printf ("%d", i);  
}
```

Für den Linker:
veränderte, eindeutige Namen

Wenn man das nicht will:
extern "C" { ... }

Optionale Parameter:

```
void print (const char *s = "\n")
```

```
{  
    printf ("%s", s);  
}
```

wird vom Compiler erledigt