

Einführung in C++, Namespaces und Ein-/Ausgabe

Sebastian Stabinger

SS2020

Hintergründe von C++

- 1979 Arbeit an "C with classes" beginnt
- 1983 Umbenennung in C++
- 1998 Erster ISO C++ Standard
- 2011 Neuer ISO C++ Standard
- 2014 Minor Revision
- 2017 Major Revision
- 2020 Major Revision



Figure: Bjarne Stroustrup. Erster Entwickler von C++

Abhängigkeiten

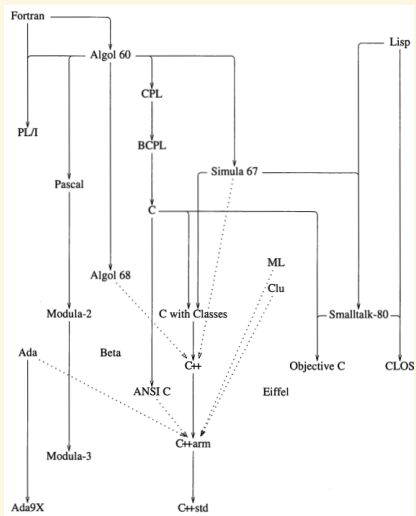


Figure: Abhängigkeiten von C++

Warum C++?

- Sehr schnell
- Hardwarenahe Programmierung ist möglich
- Problemloses verwenden von C-Libraries
- Flexibel: Vom Microcontroller über Spiele bis zum High Performance Cluster.
- Im Gegensatz zu C ist auch ein hohes Maß an Abstraktion möglich (OOP, Templates, Meta Template Programming, ...). Das macht das Programmieren letztlich einfacher.
- Grössere Standardbibliothek (aber immer noch klein im Vergleich zu Java, Python, C#, etc.) (Container, `all_of`, `find`, ...)
- Existiert seit 35 Jahren und wird auch weiterhin seinen Platz haben
- Große Code Base (Fast alle Anwendungsprogramme sind z.B. in C++ geschrieben)

Was können wir durchmachen?

- C++ ist wesentlich komplexer als C
 - C Standardwerk: 270 Seiten
 - C++ Standardwerk: 1400 Seiten!
- Wenn man C++ so programmiert wie man es Heute verwenden sollte, hat es nicht mehr sehr viel mit C zu tun.
 - In sehr vielen Firmen wird allerdings auch kein modernes C++ programmiert!
- Wir werden nur Zeit haben uns ein paar Konzepte von C++ anzusehen und für den Rest werden wir weiter C verwenden!
 - Das ist nicht ideal, aber für mehr reicht die Zeit leider nicht aus

Zur Erinnerung: Ein Hello World Programm in C:

C

```
#include <stdio.h>
int main() {
    printf("Hello World\n");
}
```

Hier das äquivalente Programm in C++:

C++

```
#include <iostream>
int main() {
    std::cout << "Hello World" << std::endl;
}
```

Ein und Ausgabe

cout (character output)

- Verwendbar nach `#include <iostream>`
- Dient der **Ausgabe** von Daten auf dem Bildschirm:
`std::cout << <daten>;`
- `cout` gibt einen Wert **automatisch korrekt** aus. `%d`, `%f`, ... ist nicht notwendig.
- Ausgaben können durch wiederholtes `<<` aneinandergereiht werden.
`std::cout << "Hello " << "World" << "!\n";`
- `std::endl` fügt einen Zeilenumbruch ein. Als Alternative kann auch `\n` wie in C verwendet werden

Beispiel

```
int a = 23; std::string b = "Bla"; double c = 42.47; char d = 'Z';
std::complex<int> e = {4, 2};

std::cout << a << " " << b << " " << c << " " << d << " " << e;
// Ausgabe: 23 Bla 42.47 Z (4,2)
```

cin (character input)

- Verwendbar nach `#include <iostream>`
- Dient der **Eingabe** von Daten durch den Benutzer:
`std::cin » <daten>;`
- Weiß wie `cout` welcher Datentyp eingelesen werden soll

Beispiel

```
int a; std::string b; double c; char d; std::complex<double> e;  
std::cin >> a; // Liest Ganzzahl  
std::cin >> b; // Liest String (ohne Leerzeichen)  
std::cin >> c; // Liest Fließkommazahl  
std::cin >> d; // Liest einzelnes Zeichen  
std::cin >> e; // Liest Komplexe Zahl im Format (real, imag)
```

Einlesen einer ganzen Zeile in einen String mit Leerzeichen etc.

```
std::string str;  
std::getline(std::cin, str);
```

Eingabebuffer löschen

```
#include <limits>  
// ...  
std::cin.clear();  
std::cin.ignore(std::numeric_limits<std::streamsize>::max())
```

cin/cout - Beispiel

```
#include <iostream>
#include <string>

int main() {
    std::string vorname;
    std::cout << "Vorname: ";
    std::cin >> vorname;
    std::string nachname;
    std::cout << "Nachname: ";
    std::cin >> nachname;
    int wiederholungen;
    std::cout << "Wiederholungen: ";
    std::cin >> wiederholungen;

    for (int i = 0; i < wiederholungen; i++) {
        std::cout << "Hallo " << vorname << " " << nachname << std::endl;
    }
}
```

Namespaces

Namespace - Das Problem in C

Namespace auf Deutsch: Namensraum ¹

Beispiel

- Angenommen wir haben zwei Bibliotheken (`control.h` und `cpu.h`)
- Beide implementieren eine Funktion `int check_state();`

```
#include "control.h"  
#include "cpu.h"  
  
if (check_state() == 42) {  
    printf("Status ok\n");  
}
```

Dieser Code wird **nicht funktionieren**. Woher soll der Compiler wissen, welche der beiden Funktionen `check_state` aufgerufen werden soll?

¹<https://de.wikipedia.org/wiki/Namensraum>

Namespace - Die Lösung in C++

- C++ löst dieses Problem durch die Einführung sogenannter **Namespaces** (Namensräume)
- Namespaces definieren **benannte Gültigkeitsbereiche** von Funktionen, Variablen, Klassen, ...
- Um von außerhalb eines Namespaces auf dessen Inhalt zugreifen zu können muss dessen **Name explizit genannt** werden.

Das vorherige Beispiel könnte gelöst werden indem die beiden Funktionen in unterschiedlichen Namespaces deklariert werden

Beispiel: Lösung in C++

```
#include "control.h" // Verwendet z.B. den Namespace ccont
#include "cpu.h" // Verwendet z.B. den Namespace cpu_n

if (cpu_n::check_state() == 42) { std::cout << "CPU ok\n"; }
if (ccont::check_state() == 47) { std::cout << "Controller ok\n"; }
```

Namespace - Deklaration

Die Deklaration eines Namespaces folgt dem Muster:

```
namespace <name> { <code> }
```

Beispiel

```
namespace ccont {  
int ok = 47;  
int check_state() { return ok; } // Unser Controller is immer OK :-)  
} // namespace ccont
```

Die Variable `ok` und die Funktion `check_state` befinden sich somit im Namensraum `ccont`

- Ein Namespace muss nicht am Stück deklariert werden. Wir können jederzeit durch einen neuen `namespace` Block einem bereits existierenden Namensraum **Elemente hinzufügen**. Auch in **unterschiedlichen Dateien**!

Namespace - Zugriff

Auf Elemente eines Namensraums wird folgendermaßen zugegriffen:

`<namespace>::<element>`

Beispiel

Wir wollen ausserhalb von `ccont` auf die Funktion `check_status()` zugreifen.

```
std::cout << ccont::check_state(); // Gibt 47 zurück
```

Standardnamensraum

Alle Standardfunktionen von C++ liegen im Namensraum `std` (für Standard).

Namespace - Import

Manchmal möchte man den Namespace nicht immer explizit angeben (besonders beim `std` Namensraum üblich)

`using`

```
std::cout << "Hello World!" << std::endl;  
using std::cout; // Ab hier können wir einfach cout verwenden  
cout << "Hello Simple World!" << std::endl;
```

`using namespace`

```
using namespace std; // Ab hier können wir überall std:: weglassen  
cout << "Hello Really Simple World!" << endl;
```

- `using` importiert ein Element
- `using namespace` importiert den ganzen Namensraum
- Man muss mit `using namespace` vorsichtig sein, da man die ganzen Vorteile von Namespaces verliert!

Namespace - Verschachtelung

Ein Namespace kann innerhalb eines anderen Namespaces deklariert werden. Wir greifen auf verschachtelte Namensräume durch wiederholtes `::` zu. Z.b. `ns1::ns2::ns3::function()`.

Beispiel

```
#include <iostream>

namespace ns1 {
    int f(int a) { return 2 * a; }
    namespace ns2 {
        int f(int a) { return a * a; }
    }
}

int main() {
    std::cout << "ns1::f(5) = " << ns1::f(5) << " ";
    std::cout << "ns1::ns2::f(5) = " << ns1::ns2::f(5) << std::endl;
    // Ausgabe: ns1::f(5) = 10 ns1::ns2::f(5) = 25
}
```