

Programmieren in C++

Christian Lang (Lac)

13. September 2019

C Grundlagen

- Vergleich C und C++ und Java
- Hauptprogramm
- Kontrollstrukturen
- Typen
- Standard-Library
- Eigene Typen definieren
- Memory Management
- Funktionen
- Makros

Vergleich C und C++

C

- kompiliert zu Binärdatei
- Präprozessor
- primitive Typen
- schwach typisiert
- Prozedural
- Pointer
- structs
- Function-Pointer
- Standard-Library

C++

- kompiliert zu Binärdatei
- Präprozessor
- primitive Typen
- **stark** typisiert
- Prozedural, **funktional**
- Pointer & **Referenzen**
- Klassen, **Vererbung**, **Polymorphie**
- Function-Pointer, **Lambdas**
- Standard-Library mit **Container** & **Algorithmen** & **etc.**
- **Namespaces**, **Exceptions**
- **Templates**

- Binärdatei
- keine VM nötig
- hohe Performanz
- C/C++ Runtime (`libc` und `libc++`)
- hohe Plattformabhängigkeit
- direkter Zugriff auf System und Speicher
- kein Garbage-Collector
- dafür Smart-Pointer mit Reference-Counting

*.c	C-Source-Files. Implementation von Funktionen
*.h	C/C++-Header-Files. Enthält mehrfach benötigte Deklarationen (struct, globale Variablen) und wird nicht direkt kompiliert sondern in Source inkludiert.
*.cpp	C++-Source-Files. Implementation von Funktionen, Methoden, etc.
*.hpp	C++-Header-Files. Wird heute nur noch für Header-Only -Implementationen verwendet.

Hauptprogramm

```
1  int main(int argc, char* argv[]) {  
2      ...  
3  }
```

argc

Anzahl der Programm-Parameter.

argv

Array mit den Programm-Parametern als C-Strings. Der erste Parameter ist immer der Pfad zum aufgerufenen Executable.

Die Parameter können auch weggelassen werden.

Scope

```
1 {  
2   ...  
3 }
```

Conditional

```
1 if (...) {  
2   ...  
3 } else if (...) {  
4   ...  
5 } else {  
6   ...  
7 }
```

While-Loop

```
1 int i = 5;  
2 while (i > 0) {  
3   i--;  
4 }
```

Do-While-Loop

```
1 int i = 5;  
2 do {  
3   i--;  
4 } while (i > 0);
```


For-Loop

```
1  for (int i = 0; i < 5; ++i) {  
2      ...  
3  }
```

Switch-Case

```
1  switch (n) {  
2      case 1:  
3          ...  
4          break;  
5      case 2:  
6          return ...;  
7      default: {  
8          ...  
9      }  
10     break;  
11 }
```

Primitive Typen

Typ	x86	x64	Bemerkung
bool	1 Byte	1 Byte	<code>!= 0</code> → <code>true</code>
char	1 Byte	1 Byte	
short	2 Byte	2 Byte	
int	4 Byte	4 Byte	
long	4 Byte	8 Byte	
long long	8 Byte	8 Byte	
float	4 Byte	4 Byte	
double	8 Byte	8 Byte	
long double	10 Byte	10 Byte	

Der Modifier `unsigned` kann bei allen Integeren als `prefix` oder `inplace` verwendet werden.

Fixed width Typen

Typ	Size
uint8_t	1 Byte
int8_t	1 Byte
uint16_t	2 Byte
int16_t	2 Byte
uint32_t	4 Byte
int32_t	4 Byte
uint64_t	8 Byte
int64_t	8 Byte
size_t	unsigned max int
uintptr_t	unsigned Pointer (max int)
intptr_t	signed Pointer (max int)

Diese und weitere sind bereits definiert in: `stdint.h` / `stddef.h`

```
1 // C style
2 #include <stdint.h>
3
4 // C++ style
5 #include <cstdint>
```

- C sowie auch C++ bringen eine **Standard-Library** mit, welche Hilfs-Typen und Funktionen enthalten.
- Als Erweiterung von C hat auch C++ **Zugriff auf alle C-Header**.
- C-Header in der Standard-Library sind mit **.h** gespeichert.
- In einem C++-Projekt werden die Pendants **ohne Extension** dafür mit **vorangehendem 'c'** verwendet.
- Die C++ Pendants kapseln die Funktionen etc. in den **Namespace std**.

```
1  int manual[4];  
2  int automatic[] = { 1, 2, 3 }  
3  
4  int third = automatic[2];  
5  manual[3] = automatic[0];
```

- Nur **statische** Arrays möglich.
- Nur auf dem **Stack** möglich.
- Deklaration in **Form**: <type> <name>[<size>]
- <size> kann weggelassen werden, wenn direkt initialisiert.
- Auch **Arrays** von **nicht-primitiven** Typen möglich.

```
1  enum Color {  
2      blau,  
3      gelb = 5,  
4      gruen,  
5  };  
6  
7  Color c = blau;  
8  if (c != gruen) {  
9      ...  
10 }
```

- Der ersten Konstante wird by-default 0 zugewiesen.
- Nachfolgende Werte werden inkrementiert.
- Jeder Konstante kann manuell ein Wert zugewiesen werden.
- Werte können auch doppelt vergeben werden.
- Konstanten sind im umgebenden Scope sichtbar.

struct

```
1  struct Point {
2      int x;
3      int y;
4      int z;
5  };
6
7  Point p;
8  p.x = 1;
9  p.y = 2;
10 int tmp = p.x;
11
12 Point o;
13 o = p;
```

- Strukturen in C vereinen Daten-Felder als **Member**-Variablen.
- Zugriff auf Member über den **.-Operator**
- Variablen mit demselben Typ können zugewiesen werden.
- Strukturen können nicht nur primitive Typen enthalten.
- Der Speicherbedarf ist **mindestens die Summe** aller Member.

```
1  union SharedData {  
2      float value;  
3      uint8_t raw[4];  
4  };  
5  
6  SharedData data;  
7  data.value = 3.1415;  
8  
9  uint8_t byte1  
10     = data.raw[0];
```

- Fast gleich wie struct, die Member **teilen** aber den Speicher.
- Es kann somit immer nur **ein Member** gesetzt werden.
- Gelesen kann aber ohne Gefahr von allen.
- Unions können nicht nur primitive Typen enthalten.
- Der Speicherbedarf ist gleich dem **grössten** Member.


```
1 // bereits definiert in stdint.h
2 typedef unsigned short uint16_t;
3
4 uint16_t var;
5 var = 65000;
```

- Definieren von Typ-**Aliasen**
- Kein **echter** neuer Typ
- **Letztes** Token definiert den Namen des neuen Alias

```
1  // stack
2  int a[4];
3  a[0] = 1;
4
5  // heap
6  int* b = malloc(4 * sizeof(int));
7  assert(b != nullptr);
8  b[0] = 1;
9  free(b);    // not automatic
```

- Stack und Heap
- Statisch → Stack
- Dynamisch → Heap

```
1  // func.h
2  int CalcSum(int a, int b);
3
4  // func.cpp
5  int CalcSum(int a, int b) {
6      return a + b;
7  }
8
9  // main.cpp
10 #include "func.h"
11 int main() {
12     return CalcSum(1, 3);
13 }
```

- Deklaration in Header
- Definition / Implementation in Source-Datei
- Inkludieren vor Verwendung

```
1  #define PRINT_INT(x) std::printf("%i", (x))
2
3  // Verwendung
4  PRINT_INT(12);
5
6  // Compiler-Intern
7  std::printf("%i", 12);
```

- Text-Ersetzung
- wird durch den Präprozessor durchgeführt
- kann schlecht gewartet und debugged werden
- kann nicht-offensichtliche Effekte haben