

# Die Programmiersprache C

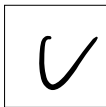
- ▶ Programmiersprache mit
  - ▶ einem höheren Abstraktionsniveau als Assembly
  - ▶ und mehr Nähe zur Hardware als (z.B.) Java
- ▶ Entwickelt in den 1970er Jahren
- ▶ Syntaktische Ähnlichkeit zu Java
  - ▶ Java basiert auf C
- ▶ Imperativ und prozedural
  - ▶ nicht objektorientiert

# Der C Standard

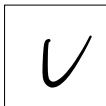
- ▶ C ist standardisiert
  - ▶ Wird seit 1990 kontinuierlich weiterentwickelt
  - ▶ Dieses Video bezieht sich auf C17 (2017)
- ▶ Definiert Anforderungen an konkrete Implementierung des Standards
  - ▶ Möglichst rückwärtskompatibel
- ▶ Konkrete Implementierung umfasst
  - ▶ Compiler
  - ▶ Standardbibliothek
  - ▶ Betriebssystem
  - ▶ und Hardware (Prozessor)
- ▶ Unterscheidung von
  - ▶ durch den Standard definiertes Verhalten
  - ▶ und “implementation-defined behavior”

## Quiz: Der C Standard (1)

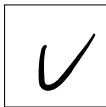
Welche Eigenschaften sind implementation-defined?



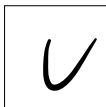
Die Anzahl der Bits in einem Byte



Die Zeichencodierung von C-Quellcode



Anforderungen an valide Dateinamen



Die Größe einer Speicheradresse

## Quiz: Der C Standard (2)

Auf welchen C-Standard bezieht sich dieses Video?

☐

C89

☐

C99

☐

C11

☒

C17

# Grundlegende Datentypen: Integer

Bezeichner	Übliche Größe (LP64)	Garantierte Größe (Standard)
_Bool	8 Bit (1 Bit nutzbar)	$\geq 1$ Bit (1 Bit nutzbar)
char	8 Bit	$\geq$ _Bool und $\geq 8$ Bit
short (int)	16 Bit	$\geq$ char und $\geq 16$ Bit
int	32 Bit	$\geq$ short und $\geq 16$ Bit
long (int)	64 Bit	$\geq$ int und $\geq 32$ Bit
long long (int)	64 Bit	$\geq$ long und $\geq 64$ Bit

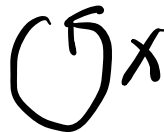
► Größe char := 1 Byte

# Grundlegende Datentypen: Integer

- ▶ Datentypen standardmäßig vorzeichenbehaftet
  - ▶ Außer char (implementation-defined) und \_Bool
- ▶ Vorzeichenlose Zahlen haben größeren positiven Wertebereich
- ▶ Overflows nur für vorzeichenlose Zahlen definiert

```
1 unsigned long l = 42;  
2 signed char c = -1;  
3 unsigned i = UINT_MAX;
```

vorzeichenbehaftet  
nicht definiert



## Quiz: Integer (1)

Was ist der Wert von b nach dem Statement `_Bool b = 42; ?`

ungleich 0

=> automatisch 1

☐

42

☒

1

☐

0

☐

Es kommt zu einem Compilerfehler

## Quiz: Integer (2)

Was ist an folgendem Codeausschnitt problematisch?

<sup>有符号</sup>  
1 `int i = INT_MAX + 1;`  
2 `unsigned u = UINT_MAX + 1;` **nicht definiert**

☐

Es gibt kein Problem

☐

Overflows bei vorzeichenlosen Zahlen sind nicht definiert

☒

Overflows bei vorzeichenbehafteten Zahlen sind nicht definiert

☐

Es kommt zu einem Compilerfehler



## Quiz: Integer (3)

Welche Kombinationen an Datentypengrößen sind valide?

☐

$\geq 8$   $\geq 16$   $\geq 32$   
char 7 Bit – int 21 Bit – long 49 Bit

☒

char 8 Bit – int 16 Bit – long 32 Bit

☒

char 8 Bit – int 32 Bit – long 32 Bit

☒

char 12 Bit – int 24 Bit – long 36 Bit

# Grundlegende Datentypen: Floating-Point

Bezeichner	Übliche Größe (LP64)
float	32 Bit
double	64 Bit

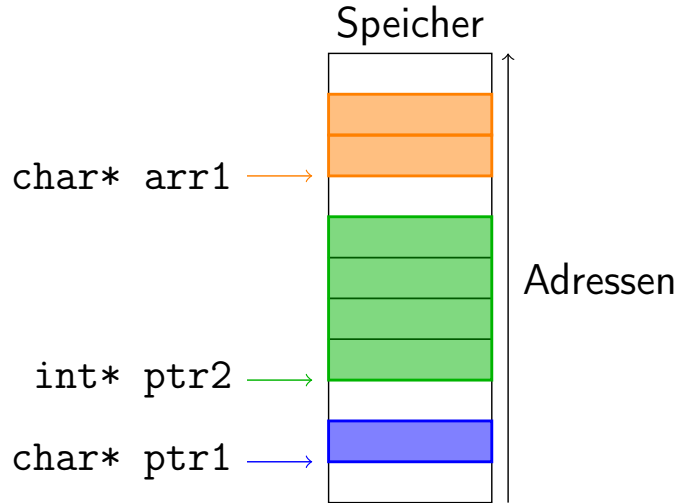
- ▶ Zudem: komplexe Zahlen (\_Complex)

# Grundlegende Datentypen: void

- ▶ void: leerer Datentyp
  - ▶ Z.B. "kein Rückgabewert" oder "keine Parameter"

A

# Pointer-Datentypen



- ▶ `void*` ist "Platzhalter" für beliebigen Pointer  
Aber: nicht direkt verwendbar 不可读!

# Funktionen in C

- ▶ Enthalten ausführbare Programmlogik
- ▶ Müssen vor Aufruf deklariert und definiert werden
  - ▶ Deklaration und Definition kann kombiniert werden

```
1 void foo(int n);           // <-- Deklaration
2 void foo(int n) {         // <-- Definition
3     ...
4 }
5
6 void bar(unsigned n) {    // <-- Deklaration + Definition
7     ...
8 }
```

# Funktionen in C

## Deklaration von Funktionen ohne Parameter

- Deklaration von Funktionen ohne Parameter: void als Parameterliste

```
1 int foo(void); // <-- RICHTIG: akzeptiert keine Parameter
2
3 int bar();      // <-- FALSCH: kann mit beliebigen Parametern
4                // definiert/aufgerufen werden
```

# Funktionen in C

## Verlassen von Funktionen

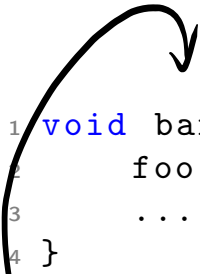
- ▶ Mittels `return` und Rückgabewert
- ▶ `void`-Funktionen haben keinen Rückgabewert
  - ▶ `return` an deren Ende optional *return;*

```
1 void foo(unsigned n, short s) {  
2     ...  
3     return; // <-- kein Rückgabewert; hier optional  
4 }  
5  
6 int bar(long long multi_word_parameter) {  
7     ...  
8     return -42; // <-- int als Rückgabewert  
9 }
```

## Quiz: Funktionen (1)

Was ist an folgendem Codeausschnitt problematisch?

```
1 void bar(void) {  
2     foo();  
3     ...  
4 }  
5  
6 void foo() {  
7     ...  
8 }
```



still allowed  
but doesn't  
mean no parameter

☐

Nichts, alles in Ordnung

☐

Die Funktion foo wurde ohne Angabe von Parametern bzw. void deklariert

☒

Die Funktion foo wird genutzt, bevor sie deklariert wird

☐

Es kommt zu einem Compilerfehler

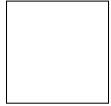


## Quiz: Funktionen (2)

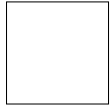
Wie viele Parameter nimmt die deklarierte Funktion `void foo();` ?



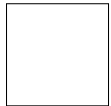
Beliebig viele



Keine



Das ist undefiniertes Verhalten



Es kommt zu einem Compilerfehler

# Die main-Funktion

- ▶ Eintrittspunkt des Programms
  - ▶ Rückgabewert: Exit Code
    - ▶ Standardisierte Konstanten EXIT\_SUCCESS und EXIT\_FAILURE
- C17*  
*0*                      *1*

```
1 int main(void) {  
2     ...  
3     return EXIT_SUCCESS;  
4 }
```

```
1 int main(int argc, const char** argv) {  
2     ...  
3     return 1; // Implementation-defined error code  
4 }
```

# Variablen in C

- ▶ Variable muss vor Nutzung deklariert werden
  - ▶ Alloziert Speicherplatz für diese
- ▶ Wert bis zur Zuweisung undefiniert
  - ▶ Kann mit Deklaration kombiniert werden

```
1 TYPE NAME [= VALUE];           // Deklaration [und Zuweisung]
2
3 const TYPE NAME = VALUE;       // Deklaration und Zuweisung einer
4                                // konstanten Variable
                                " final "
```

# Variablen in C

## Pointer und const

Vorsicht bei const in Kombination mit Pointern:

```
1 const TYPE* PTR [= ADDR];           // Pointer auf konstante Daten
2
3 TYPE* const PTR = ADDR;             // Konstanter Pointer auf
4                                     // variable Daten
5
6 const TYPE* const PTR = ADDR;       // Konstanter Pointer auf
7                                     // konstante Daten
```

Pointer Daten

const int\* PTR; Pointer 指向 const int

int\* const PTR=ADDR; const 的 PTR

const int\* const PTR=ADDR; 指向 ADDR

# Variablen in C

## Scopes 范围

```
1 void foo() {  
2     int a = 42;  
3 }  
4  
5 void bar() {  
6     int b = a; // FEHLER: a ist nur in foo sichtbar  
7  
8     {  
9         int c = b; // OK  
10    }  
11  
12    int d = b; // OK  
13    int e = c; // FEHLER: c ist hier nicht mehr sichtbar  
14 }
```

nicht erreichbar

但 construct { } 是可见的

nicht erreichbar

# Variablen in C

## Zuweisung von konstanten Werten

```
1  int i;  
2  i = -2;           // negative Konstante im Dezimalsystem  
3  i = 0xDEADBEEF;   // Konstante im Hexadezimalsystem  
4  i = 011;          // Konstante im Oktalsystem (führende Null!!)  
5  i = 'A';          // "character literal" - hier wird automatisch  
6                    // der entsprechende numerische Wert für den  
7                    // Buchstaben "A" eingefügt.  
8                    // Siehe man 7 ascii für eine Tabelle.  
9                    (linux commando)  
10 double d;  
11 d = 2.0; (implizit) // double-Konstante  
12 d = 2.0f;          // float-Konstante
```

Δ

## Quiz: Variablen (1)

Was gibt die Funktion foo zurück?

```
1 int foo(void) {  
2     int a;  
3     return a;  
4 }
```

☒

*a hat keinen Wert zugewiesen bekommen.*  
Einen undefinierten Wert

☐

0

☐

*undefiniert ≠ zufällig*  
Einen zufälligen Wert

☐

*nein!*  
Es kommt zu einem Compilerfehler

## Quiz: Variablen (2)

Welchen Wert (dezimal) hat `i` nach dem Statement `int i = 2.0` ?  
**Δ**

☐

2.0

☒

2

*umgewandelt*

☐

Einen undefinierten Wert

☐

Es kommt zu einem Compilerfehler



## Quiz: Variablen (3)

Welchen Wert (dezimal) hat j nach dem Statement `int j = -042` ?

☐

-42

☐

-66

☒

-34

☐

Es kommt zu einem Compilerfehler

## Quiz: Variablen (4)

Welchen Wert (dezimal) hat c nach dem Statement `char c = 't' ?`

☒

116

☐

74 (Hex)

☐

84

☐

Es kommt zu einem Compilerfehler

## Quiz: Variablen (5)

Welchen Wert (dezimal) hat a nach dem folgenden Codeausschnitt?

```
1 const int a;  
2 a = 't' - 0x42;
```

Const 3. 4. 5.  
in Deklaration (同時!)  
sich mit einem  
Wert zuweisen

☐

50

☐

82

☐

Einen undefinierten Wert

☒

Es kommt zu einem Compilerfehler

## Quiz: Variablen (6)

Was passiert, wenn in einer Funktion eine Variable `int a;` deklariert wird und sie verwendet wird, bevor ihr ein Wert zugewiesen wurde?

☐

Es kommt zu einem Compilerfehler

☒

Es kann zu einem Segmentation Fault kommen

☒

Der Wert von `a` ist undefiniert

☐

Der Wert von `a` wird automatisch zu 0 initialisiert

202020

# Einige arithmetische und logische Operationen

Operation (unsigned a = 42;)	Operation (direkte Zuweisung)	Bedeutung	Ergebnis (dezimal)	Ergebnis-Typ
$a = a + 42;$	$a += 42;$	Addition	84	unsigned
$a = a - 42;$	$a -= 42;$	Subtraktion	0	unsigned
$a = a * 42;$	$a *= 42;$	Multiplikation	1764	unsigned
$a = a / 5;$	$a /= 42;$	Division	8	unsigned
$a = a \% 5;$	$a \% = 42;$	Modulo	2	unsigned
$a = a \&\& 0;$	000000	logisches UND	0	int
$a = a    0;$	-	logisches ODER	1	int
$a = !a;$	-	logisches NOT	0	int
$a = a << 2;$	$a <<= 2;$	Linksshift 去除	168 10101000	unsigned
$a = a >> 2;$	$a >>= 2;$	Rechtsshift 移0	10 1010	unsigned
$a = a \& 0x3;$	$a \&= 0x3$	bitweises UND	2	unsigned
$a = a   0x5;$	$a  = 0x5$	bitweises ODER	47	unsigned
$a = a \wedge 0xff;$	$a \wedge= 0xff$	bitweises XOR	213	unsigned
$a = \sim a;$	-	bitweises NOT	4294967253	unsigned

$2 + 4 + 16 = 21$

视为  
int的  
"1"  
(book)

242 .. 0  
221 .. 2  
210 .. 0  
215 .. 2  
212 .. 0  
101010  
010101

42  
84  
168  
1764

42  
2.  
或  
000000

去除  
移0  
10101000  
1010

## Quiz: Logische Operatoren

Welchen Datentyp hat der Ausdruck  
`a && b` (für ein `short` `a` und ein `long` `b`)?

☒

`int`

☐

`short`

☐

`long`

☐

Es kommt zu einem Compilerfehler

# Kontrollflussstrukturen

## if-else Bedingungen

if - else  
最好用 if - else if

```
1 if (x > 2.4) {  
2     ...  
3 } else if (x < 0x123456789) { // else if branch optional  
4     ...  
5 } else { // else branch optional  
6     ...  
7 }
```

# Kontrollflussstrukturen

## while und do-while Schleifen

```
1 while (n-- > 0) {  
2     ...  
3     if (x == y) {  
4         break;  
5     }  
6     ...  
7 }
```

```
1 do {  
2     ...  
3     if (x == y) {  
4         continue;  
5     }  
6     ...  
7 } while (--n > 0);
```



# Kontrollflussstrukturen

## for Schleifen

```
1 // Variante 0
2 for (int i = 0; i < 42; i++) { ... }
3
4 // Variante 1
5 for (int i = 0, j = 0; ...) { ... }
6
7 // Variante 2
8 int k;
9 for (k = 0; k < 42; k++) { ... }
10
11 // Variante 3
12 for (;;) { ... } // = while (true) while (1) { ... }
13
14 // Variante 4
15 for (unsigned i = n; i-- > 0; ) { ... }
```

# Kontrollflussstrukturen

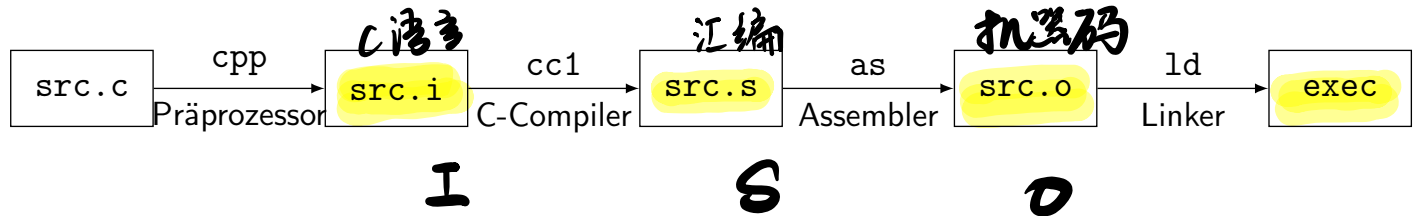
## switch Statements

```
1  switch (x) {  
2      case -42:  
3          ...  
4          break;  
5      case 'A':  
6          ...  
7          /* fall through */  
8      case 'B':  
9          ...  
10         break;  
11     default:  $\Delta$   
12         ...  
13         break;  
14 }
```

weiter (Bug)

# Der C-Präprozessor

- ▶ Vor dem Kompilieren: *Preprocessing*  $\Delta$
- ▶ Auflösen von Makros
- ▶ Kombination mehrerer Dateien



# Der C-Präprozessor

## Makros



```
1 #define NUMBER 42      // Ersetze NUMBER durch 42
2 #define MYNUM 2 + 3    // Ersetze MYNUM durch 2 + 3
3
4 int a = NUMBER;        // = 42
5 int b = MYNUM * 2;     // = 2 + 3 * 2 = 8 (nicht (!) 10)
6
7 #undef MYNUM
```

# Der C-Präprozessor

## if-else Konstrukte

```
1 #define MYFLAG 0
2
3 #if MYFLAG
4 const char c = 'A';
5 #else
6 const char c = 'B';
7 #endif
8
9 #if 0
10 int x = 42;    // auskommentierter Code
11 #endif
```

# Der C-Präprozessor

## #include Direktiven

```
1 #include <system_header.h> // Copy-paste Inhalte von
2                             // system_header.h an diese Stelle
3
4 #include "local_header.h"  // Copy-paste Inhalte von
5                             // local_header.h an diese Stelle
```

↓ C-Standard

↑ eigene

# Header-Dateien

foo.h: <sup>↑ header</sup>

```
1 void foo(void);
```

foo.c:

```
1 #include "foo.h"
2
3 void foo(void) {
4     ...
5 }
```

具体实现  
不写在header  
里面！

main.c:

```
1 #include "foo.h"
2
3 int main(void) {
4     foo();
5     return 0;
6 }
```

# Sichtbarkeit

foo.h:

```
1 void func(void);
```

foo.c:

```
1 #include "foo.h"
```

```
2
```

```
3 static void helper(void) {
```

```
4     ...
```

```
5 }
```

```
6
```

```
7 void func(void) {
```

```
8     ...
```

```
9 }
```

main.c:

```
1 #include "foo.h"
```

```
2
```

```
3 static void helper(void) {
```

```
4     ...
```

```
5 }
```

```
6
```

```
7 int main(void) {
```

```
8     func();
```

```
9     return 0;
```

```
10 }
```



# Standard-Header

## ► Nutzung der Standardbibliothek:

- Kein "Import-System"
- Sondern über Header



stdio.h  
string.h  
stddef.h

Standard

```
1 // Systemweite Bibliotheksheader
2 #include <stdio.h>    // Input-Output Funktionalität
3 #include <string.h>   // Funktionen zur Stringmanipulation
4
5 #include <stddef.h>   // Definiert u.a. size_t (unsigned Typ,
6                       // max. Grösse von Objekten im Speicher).
7                       // Bereits indirekt durch stdio.h
8                       // eingebunden.
9
10 // Lokaler Header des Projekts
11 #include "myheader.h"
```

# Standard-Header

stdint.h und stdbool.h

- ▶ stdint.h definiert fixed-width Integer Typ

Bezeichner	Übliche Größe (LP64)	Garantierte Größe (Standard)
_Bool	8 Bit (1 Bit nutzbar)	≥ 1 Bit (1 Bit nutzbar)
char	8 Bit	≥ _Bool und ≥ 8 Bit
short (int)	16 Bit	≥ char und ≥ 16 Bit
int	32 Bit	≥ short und ≥ 16 Bit
long (int)	64 Bit	≥ int und ≥ 32 Bit
long long (int)	64 Bit	≥ long und ≥ 64 Bit

规定  
int 的大小

Signed	Unsigned	Größe
int8_t	uint8_t	8 Bit
int16_t	uint16_t	16 Bit
int32_t	uint32_t	32 Bit
int64_t	uint64_t	64 Bit

- ▶ stdbool.h enthält syntaktischen Zucker für boolsche Werte
  - ▶ bool als Synonym für \_Bool
  - ▶ true und false als Synonyme für die Integer Konstanten 1 und 0

Δ

Δ

# printf – Beispiel

Hello World in C (mit printf):

```
1 #include <stdio.h> // <-- Wir brauchen die Deklaration von
2                       // printf
3
4 int main(void) {
5     // Schreibe "Hello World!" gefolgt von einer Newline
6     printf("Hello World!\n");
7
8     return 0;
9 }
```

*String "..."*

*"\n": Newline*

# printf – Format Strings

- ▶ printf bietet vielfältige Ausgabemöglichkeiten
  - ▶ Funktionssignatur: `int printf(const char* format, ...);`
  - ▶ format ist sog. Format String

```
1 unsigned a = 0x42;  
2  
3 printf("The value of a is: %u\n", a);
```

## printf – Conversion Specifiers

Specifier	Argumenttyp	Ausgabe
d	Signed Integer	Dezimaldarstellung
u	Unsigned Integer	Dezimaldarstellung
x oder X	Unsigned Integer	Hexadezimaldarstellung
c	Signed Integer	Als ASCII-Zeichen
s	<u>const char*</u>	Als String

- ▶ Optionale Angabe eines Length Modifiers vor dem Conversion Specifier
  - ▶ Bedeutung abhängig von Conversion Specifier
  - ▶ Z.B. %ld für einen long int
- ▶ Weitere Informationen: man 3 printf und man inttypes.h

## Quiz: printf (1)

Was ist die Ausgabe, die aus folgendem printf-Aufruf resultiert (ohne Newline)?

```
printf("%u\n", -1);
```

☐

-1



4294967295 (bei 4-Byte Größe von int bzw. unsigned)

☐

Es kommt zu einem Compilerfehler

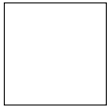
## Quiz: printf (2)

Was ist die Ausgabe, die aus folgendem printf-Aufruf resultiert (ohne Newline)?

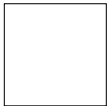
```
printf("%c\n", "test");
```



Das niedrigwertigste Byte der Adresse von "test"



t



Es kommt zu einem Compilerfehler

## Quiz: printf (3)

Was könnte bei folgender printf-Ausgabe unter Umständen anders laufen, als man es auf den ersten Blick vermuten würde?

```
printf("%s", "test");
```

☐

Ein String kann kein Argument für printf sein

☐

Ohne abschließende Newline ist der Format String invalide

☒

Es kommt möglicherweise erst einmal zu keinem Output

\\n 才立即输出  
否则进入buffer区



## Quiz: printf (4)

Welchen Datentyp sollte der Parameter x haben?

```
printf("%" PRIx64 "\n", x);
```

↓  
*Makros*

☐

unsigned long long

☐

int64\_t

☒

uint64\_t

☐

Das Programm kompiliert nicht