Praktikum C-Programmierung 2026

Basics

Jannek Squar

2026-04-15

Scientific Computing
Universität Hamburg

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Outline

Table of Contents

Genereller Aufbau eines C-Programms

```c
#include <stdio.h>            // System-Header
#include "foo.h"              // Eigene Header

#define MAX 10                // Präprozessor-Direktiven

int foo = 42;                 // Globale Variablen

void begruessung(int);        // Vorwärtsdeklarationen

int main(){                   // main
  begruessung(foo);
  return 0;
}

void begruessung(int bar){    // weitere Funktionen
  int baz = bar;              // lokale Variable
  printf("Hallo %d!\n", baz);
}
```

genereller_aufbau.c

Statements und Expressions

```
1  // Expressions
2  a + b
3  a*b / 14
4  a >= b
5  d = 10
6  c = d = 10
7
8  // Statements
9  ;
10 d = 10;
11 control_statement
12 {
13   statement;
14   statement;
15 }
```

Declaration vs. Definition vs Initialisierung

- Deklaration: Bekanntmachung eines Bezeichners
- Definition: Speicher-Reservierung bzw. Implementation
- Initialisierung: Zuweisung Startwert

```c
// Deklaration
int max(int a, int b);
extern char c;

// Definition (und Deklaration)
int max(int a, int b) {  /* ... */  }
char c;

// Initialisierung
c = 4;
```

Weiterführender Lesestoff (Link)

Präprozessor

`if`	`ifdef`	`include`	`defined`
`elif`	`ifndef`	`embed (C23)`	`__has_include (C23)`
`else`	`elifdef (C23)`	`line`	`__has_embed (C23)`
`endif`	`elifndef (C23)`	`error`	`__has_c_attribute (C23)`
	`define`	`warning (C23)`	
	`undef`	`pragma`	

C Präprozessor Keywords (Link)

Beispiele I

```c
#if __STDC__ != 1
#  error "Not a standard compliant compiler"
#endif

#if __STDC_VERSION__ >= 202311L
#  warning "Using #warning as a standard feature"
#endif

#include <stdio.h>

int main (void)
{
    printf("The compiler used conforms to the ISO C Standard !!");
}
```

warning_directive.c (Link)

```
$ gcc -E warning_directive.c > warning_directive.i
warning_directive.c:6:4: warning: #warning "Using #warning as a standard feature" [-Wcpp]
    6 | #  warning "Using #warning as a standard feature"
      |    ^~~~~~~
```

Beispiele II

```c
#define ABCD 2
#include <stdio.h>

int main(void)
{

#ifdef ABCD
  printf("1: yes\n");
#else
  printf("1: no\n");
#endif
}
```

conditional_inclusion.c (Link)

- 1: yes
- 1: no
- nix

Beispiele II

```c
#define ABCD 2
#include <stdio.h>

int main(void)
{

#ifdef ABCD
  printf("1: yes\n");
#else
  printf("1: no\n");
#endif
}
```

conditional_inclusion.c (Link)

- 1: yes ✓
- 1: no
- nix

Beispiele III

```c
#define ABCD 2
#include <stdio.h>

int main(void)
{
#ifndef ABCD
  printf("2: no1\n");
#elif ABCD == 2
  printf("2: yes\n");
#else
  printf("2: no2\n");
#endif
}
```

conditional_inclusion2.c (Link)

- 2: no1
- 2: yes
- 2: no2
- nix

Beispiele III

```c
#define ABCD 2
#include <stdio.h>

int main(void)
{
#ifndef ABCD
  printf("2: no1\n");
#elif ABCD == 2
  printf("2: yes\n");
#else
  printf("2: no2\n");
#endif
}
```

conditional_inclusion2.c (Link)

- 2: no1
- 2: yes ✓
- 2: no2
- nix

Beispiele IV

```c
#include <stdio.h>

int main(void)
{

#if defined(ABCD) && (ABCD > 0)
    printf("3: yes\n");
#else
    printf("3: no\n");
#endif
}
```

conditional_inclusion3.c (Link)

- 3: yes
- 3: no
- nix

Beispiele IV

```c
#include <stdio.h>

int main(void)
{

#if defined(ABCD) && (ABCD > 0)
    printf("3: yes\n");
#else
    printf("3: no\n");
#endif
}
```

conditional_inclusion3.c (Link)

- 3: yes (✓)
- 3: no ✓
- nix

Makrodefinition beim Kompilieren

```c
#include <stdio.h>

int main(void)
{

#if defined(ABCD) && (ABCD > 0)
    printf("3: yes\n");
#else
    printf("3: no\n");
#endif
}
```

„Normal“

```
$ gcc conditional_inclusion3.c
$ ./a.out
3: no
```

Mit -D<varname>=<value>

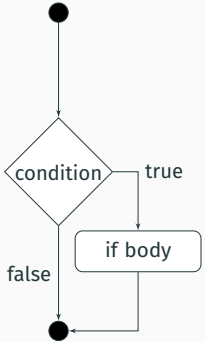
```
$ gcc -DABCD=4 conditional_inclusion3.c
$ ./a.out
3: yes
```

Table of Contents

if ...

```
1  if ( condition )
2  {
3    statement;
4  }
```

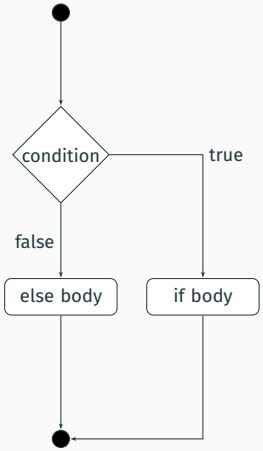
```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int answer = 42;
6
7    if ( answer == 42 )
8    {
9      printf("Here!\n");
10   }
11 }
```



if ... else ...

```
1  if ( condition )
2  {
3    statement;
4  }
5  else
6  {
7    statement;
8  }
```

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int answer = 84;
6    if ( answer == 42 )
7    {
8      printf("Here!\n");
9    }
10   else
11   {
12     printf("Alternative universe!\n");
13   }
14 }
```

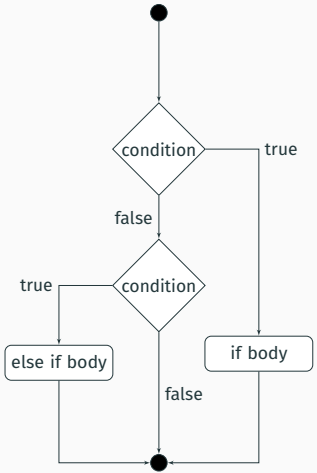


if ... else if ...

```c
if ( condition )
{
  statement;
}
else if ( condition )
{
  statement;
}
```

```c
#include <stdio.h>

int main(void)
{
  int answer = 21;
  if ( answer == 42 )
  {
    printf("Here!\n");
  }
  else if ( answer == 21 )
  {
    printf("Specific alternative universe!\n");
  }
}
```



Integer-Vergleich mit Switch

```
switch (expression)
{
  case A:
    statement;
    break;
  case B:
    statement;
    break;
  case C:
    statement;
    break;
}
```

Integer-Vergleich mit Switch

```
1  switch (expression)
2  {
3      case A:
4      case B:
5          statement;
6          break;
7      case C:
8          statement;
9                          /*FALLTHROUGH*/
10     case D:
11         statement;
12         break;
13 }
```

Integer-Vergleich mit Switch - Default

```
1     switch (expression)
2     {
3     case A:
4     case B:
5       statement;
6       break;
7     case C:
8       statement;
9                     /*FALLTHROUGH*/
10    case D:
11      statement;
12      break;
13    default:
14      statement;
15  }
```

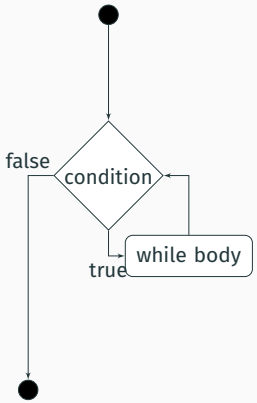
while-Schleife[1]

```c
while ( condition )
{
  statement;
  statement;
}
```

```c
#include <stdio.h>

int main(void)
{
  int i = 0;

  while ( i < 6 )
  {
    printf("%d ", i);
    i++;
  }
}
```



[1] while-loop.c

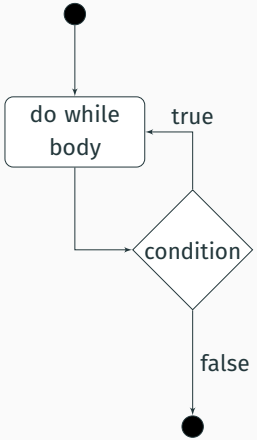
do-while-Schleife[2]

```
do
{
  statement;
  statement;
} while ( condition );
```

```c
#include <stdio.h>

int main(void)
{
  int i = 0;

  do
  {
    // will be executed at least once
    printf("%d ", i);
    i++;
  } while ( i < 1 );
}
```



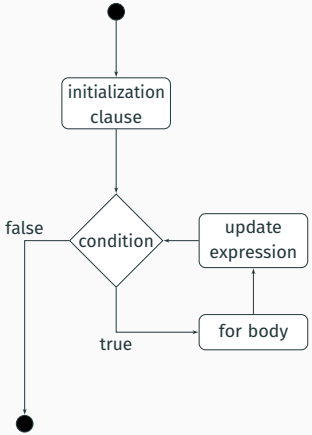
[2] do-while-loop.c

for-Schleife

```
for (clause; condition; expression)
{
  statement;
  statement;
}
```

```c
#include <stdio.h>

int main(void)
{
  for (int i = 0; i < 10; i++)
  {
    printf("%d ", i);
  }
}
```



Kontrollierte Sprünge[3]

```c
for (int i = 0; i < 10; i++)
{
  if ( i == 4 || i == 6 )
    continue;

  if ( i == 9 )
    break;

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

[3] for-loop.c

Der Vollständigkeit halber: goto[4]

```
goto LABEL;
LABEL : Anweisung;
```

```c
#include <stdio.h>

int main(void) {
   int i,j,k;
   for(i=1; i<10; i++) {
      for(j=1; j<10; j++) {
         for(k=1; k<10; k++) {
            printf("Tiefe Verschachtelungsebene\n");
            goto RAUS;
         }
      }
   }
   RAUS : printf("Mit einem Sprung raus hier \n");
   return 0;
}
```

▸ Relevantes XKCD

[4] goto.c

Table of Contents

Benennungsschema

Regeln für Bezeichner-Syntax:

- Regex: ^[a-zA-Z_][a-zA-Z0-9_]*$
  - Gültige Zeichen: Buchstaben, Ziffern, Unterstriche
  - Keine Ziffer als erstes Zeichen
  - Unterstrich als erstes Zeichen vermeiden
- Case-Sensitive
- Keine Keywords

Zugrunde liegende C-Grammatik in Backus Naur Form (Link)

Keywords in C

alignas (C23)	extern	sizeof	_Alignas (C11)(deprecated in C23)
alignof (C23)	false (C23)	static	_Alignof (C11)(deprecated in C23)
auto	float	static_assert (C23)	_Atomic (C11)
bool (C23)	for	struct	_BitInt (C23)
break	goto	switch	_Bool (C99)(deprecated in C23)
case	if	thread_local (C23)	_Complex (C99)
char	inline (C99)	true (C23)	_Decimal128 (C23)
const	int	typedef	_Decimal32 (C23)
constexpr (C23)	long	typeof (C23)	_Decimal64 (C23)
continue	nullptr (C23)	typeof_unqual (C23)	_Generic (C11)
default	register	union	_Imaginary (C99)
do	restrict (C99)	unsigned	_Noreturn (C11)(deprecated in C23)
double	return	void	_Static_assert (C11)(deprecated in C23)
else	short	volatile	_Thread_local (C11)(deprecated in C23)
enum	signed	while	

Keywords in C (Link)

Relevante Keywords

Ausgewählte Keywords:

```
// Types and Related
char     double    float     int       long      short     void
enum     union     struct    typedef
sizeof

// Modifiers/Qualifiers for Variables/Types
const    signed    unsigned    static    extern    restrict

// Function-Specific
inline   return

// Control
break    case      continue  default   do        else      for
goto     if        return    switch    while
```

Basisdatentypen

```
1  // Integer-Typen
2  char
3  short
4  int
5  long
6  long long
7
8  // Gleitkommazahlen
9  float
10 double
11 long double
```

Basisdatentypen Beispiele

```c
char a         = 'c';
int b          = 1;
short c        = 2;
short int d    = 3;
int e          = -4.8;     // -4, kein mathematisches Runden sondern truncation
float f        = 5;        // 5.0
float f        = (float)5; // expliziter Typecast
float g        = 6.0;
char string[] = "Hallo Welt";
```

Literale

Zeichenfolge zur Darstellung der Werte von Basistypen

- Ganzzahlen
  - Dezimal: `123`
  - Oktal: `0173`
  - Hexadezimal: `0x7B`
- Fließkommazahlen
- Suffixe spezifizieren genauen Datentyp
  - Bsp.: `66u` bzw. `1234567L`
- Zeichenliteral `'A'`
- *String* `"A"`, endet mit `'\0'`
  - Achtung: `'A' != "A"`

Basistypen (Auszug)

Angaben laut Standard (`<limits.h>`, `<float.h>`)

Datentyp	Bytes	Bereich
unsigned char	1	$2^8 - 1 = 255$
unsigned short int	≥ 2	$2^{16} - 1 = 65535$
unsigned int	≥ 2[5]	$2^{16} - 1 = 65535$
unsigned long int	≥ 4	$2^{32} - 1 = 4294967295$
unsigned long long int	≥ 8	$2^{64} - 1 = 18446744073709551615$
float		1E+37
double		1E+37[6]
long double		1E+37[7]

[5] Modernes System: 4 Bytes
[6] Modernes System: 1E+308
[7] Modernes System: 1E+4932

Konkrete Größe der Datentypen[8]


```c
#include <stdio.h>
#include <limits.h>
#include <float.h>
#include <stdbool.h>

int main(void) {
    printf("Type               Size (bytes)    Max value\n");
    printf("-----------------------------------------------------\n");
    printf("char               %zu              %d\n",          sizeof(char), CHAR_MAX);
    printf("signed char        %zu              %d\n",          sizeof(signed char), SCHAR_MAX);
    printf("unsigned char      %zu              %u\n",          sizeof(unsigned char), UCHAR_MAX);
    printf("short              %zu              %d\n",          sizeof(short), SHRT_MAX);
    printf("unsigned short     %zu              %u\n",          sizeof(unsigned short), USHRT_MAX);
    printf("int                %zu              %d\n",          sizeof(int), INT_MAX);
    printf("unsigned int       %zu              %u\n",          sizeof(unsigned int), UINT_MAX);
    printf("long               %zu              %ld\n",         sizeof(long), LONG_MAX);
    printf("unsigned long      %zu              %lu\n",         sizeof(unsigned long), ULONG_MAX);
    printf("long long          %zu              %lld\n",        sizeof(long long), LLONG_MAX);
    printf("unsigned long long %zu              %llu\n",        sizeof(unsigned long long), ULLONG_MAX);
    printf("float              %zu              %e\n",          sizeof(float), FLT_MAX);
    printf("double             %zu              %e\n",          sizeof(double), DBL_MAX);
    printf("long double        %zu              %Le\n",         sizeof(long double), LDBL_MAX);
    printf("bool               %zu              %d\n",          sizeof(bool), 1);
    return 0;
}
```

datentypen_limits.c

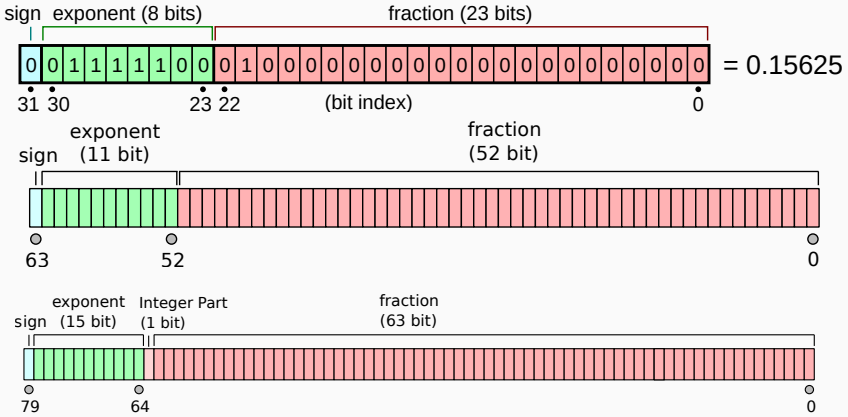
Gleitkommazahlen

„Die Norm IEEE 754 definiert Standarddarstellungen für binäre und dezimale Gleitkommazahlen in Computern und legt genaue Verfahren für die Durchführung mathematischer Operationen, insbesondere für Rundungen, fest.“ [1]

- C orientiert(!) sich an IEEE 754
- IEEE 754 definiert Formate
  - Orientierung für `float` und `double`
  - `long double` nur Empfehlung (extended precision Format)

Bit-Darstellung Gleitkommazahlen

Float [2]/Double [3]/Extended Floating Point (Long Double) [4]



Fließkommazahlen - Genauigkeit

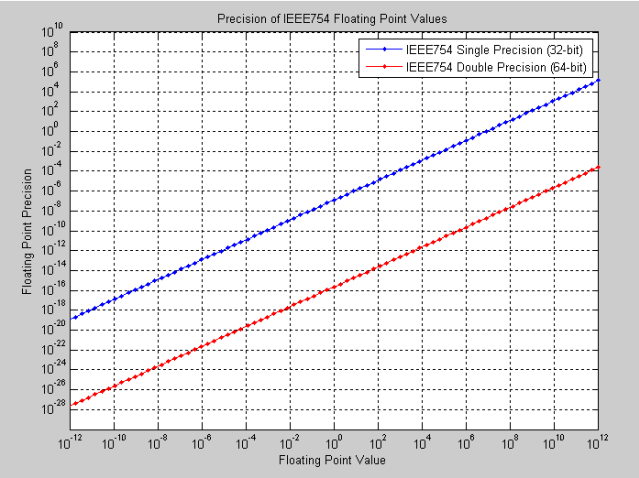


Abbildung 1: Auflösung `float` vs `double` [1]

Fließkommazahlen- Genauigkeit

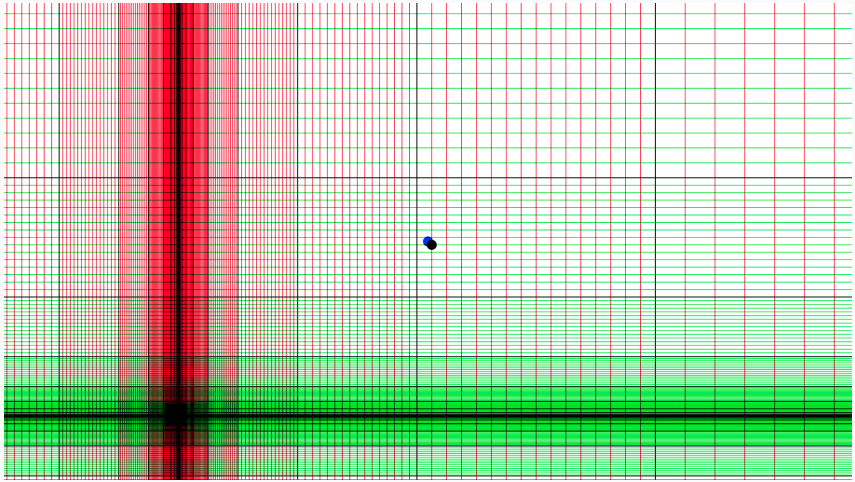


Abbildung 2: Verlust der Genauigkeit [5]

Rundungsfehler[9]

Assoziativität wegen Rundungsfehlern nicht gewährleistet!

```c
#include <stdio.h>

int main(void) {
    double a = 1e16;
    double b = -1e16;
    double c = 1.0;

    printf("(a + b) + c = %.1f\n", (a + b) + c);
    printf("a + (b + c) = %.1f\n", a + (b + c));
}
```

```
$ ./assoziativitaet
(a + b) + c = 1.0
a + (b + c) = 0.0
```

Vorsicht bei Bedingungen/Vergleichen von floats!

[9] assoziativitaet.c

Rundungsfehler[9]

Assoziativität wegen Rundungsfehlern nicht gewährleistet!

```c
#include <stdio.h>

int main(void) {
    double a = 1e16;
    double b = -1e16;
    double c = 1.0;

    printf("(a + b) + c = %.1f\n", (a + b) + c);
    printf("a + (b + c) = %.1f\n", a + (b + c));
}
```

```
$ ./assoziativitaet
(a + b) + c = 1.0
a + (b + c) = 0.0
```

Vorsicht bei Bedingungen/Vergleichen von floats!

[9] assoziativitaet.c

- Variablen gültig innerhalb Scope
  - Global
  - Lokal innerhalb geschweifter Klammern
- Garantierte Default-Initialisierung[10]:
  - Default-Initialisierung globaler Variablen
  - Default-Initialisierung „lokaler“ Variablen mit `static` Modifier
  - **Keine** Default-Initialisierung lokaler Variablen
- Empfehlung: Immer bei Deklaration auch gleich initialisieren

[10] Compiler könnte mehr initialiseren (UB)

Scope Beispiel[11]

```c
#include <stdio.h>

char  global_char;
int   global_int;
float global_float;
void  *global_ptr;

int main()
{
  char  local_char;
  int   local_int;
  {
    float local_float;
    void  *local_ptr;

    printf("\tlocal:\tglobal\n");
    printf("char \t %d \t %d\n", local_char,  global_char);
    printf("int  \t %d \t %d\n", local_int,   global_int);
    printf("float\t %f \t %f\n", local_float, global_float);
    printf("ptr  \t %p \t %p\n", local_ptr,   global_ptr);
  }
  return 0;
}
```

[11] scope.c

Scope Beispiel

Nicht-deterministische Ausgabe

```
$ ./scope
       local         global
char   0             0
int    -1042697840   0
float  0.000000      0.000000
ptr    (nil)         (nil)
```

Operatoren

Unterscheidung:

- Anzahl Operanden
  - Unär
  - Binär
  - Ternär
- Position
  - Infix
  - Präfix
  - Postfix/Suffix
- Assoziativität
  - Linksassoziativität
  - Rechtsassoziativität

Unäre & binäre Operatoren[13]

Unär

- Negation: (! Bedingung), (**not** Bedingung)[12]

Binär

- logisches UND: (Bedingung && Bedingung), (Bedingung **and** Bedingung)
- logisches ODER: (Bedingung || Bedingung), (Bedingung **or** Bedingung)
- logisches GLEICH: (Ausdruck == Ausdruck)

⚠ Verwechslungsgefahr mit |, & und =

Operatoren können beliebig verschachtelt werden

[12] Benötigt iso646.h-Header
[13] verwechslung.c

Ternärer Operator

„Normale“ Form:

```
1  if (a < b) {
2    min = a;
3  }
4  else {
5    min = b;
6  }
```

Kurze Form:

```
1  // condition ? expression : expression
2  min = (a < b) ? a : b;
```

- ⊕ praktisch
- ⊖ verminderte Lesbarkeit

Operatoren

```c
// Arithmetisch
// + - * / %
a = a - 1;

// Erweitert arithmetisch
// += -= *= /= %=
a -= 1;

//Inkrement/Dekrement
// ++ --
a++;

// Bit-Operatoren
// & | ^ ~ >> <<

// Erweiterte Bit-Operatoren
// &= |= ^= ~= >>= <<=

// sizeof Operator
sizeof(a);
sizeof(a++); // a++ wird nicht ausgeführt!
sizeof(double);
```

Inkrement/Dekrement

```c
#include <stdio.h>

int main(void) {
   int i=1;

   printf("i=%d\n",i);         // i=1
   i++;
   printf("i=%d\n",i);         // i=2
   printf("i=%d\n",i++);       // i=2
   printf("i=%d\n",i);         // i=3
   printf("i=%d\n",++i);       // i=4
   return 0;
}
```

Achtung: Mehrfache Änderung der gleichen Variable ist UB.

- Bsp: `int x = i++ + i++;`
- Bsp: `printf("%d %d\n", i, i++);`

Wertigkeit [14]

Präzedenz	Operator	Beschreibung	Assoziativität
1	++ -- () [] . -> (type){list}	Suffix/postfix Inkrement/Dekrement Funktionsaufruf Array-Zugriff Zugriff auf Struct-/Union-Member Zugriff auf Struct-/Union-Member über Pointer zusammengesetzte Literale/Compound literal(C99)	Links→rechts
2	++ -- + - ! ~ (type) * & sizeof _Alignof	Prefix Inkrement/Dekrement Unäres plus/minux Logisches/bitweises NOT Type cast Dereferenzierung Adress-Operator Size-of Anforderung Speicher-Alignment(C11)	Rechts→links
3	* / %	Multiplikation, Division, Modulo	Links→rechts
4	+ -	Addition, Subtraktion	
5	<< >>	Bitweiser links-/Rechts-Shift	
6	< <= > >=	Vergleichsoperator Vergleichsoperator	
7	== !=	Vergleichsoperator	
8	&	Bitweise AND	
9	^	Bitweise XOR	
10	|	Bitweise OR	
11	&&	Logisches AND	
12	||	Logisches OR	
13	?:	Ternärer Bedingungsoperator	Rechts→links
14	= += -= *= /= %= <<= >>= &= ^= |=	Zuweisungsoperator Zuweisung inkl. Addition/Subtraktion Zuweisung inkl. Multiplikation/Division/Modulo Zuweisung inkl. bitweisem Links-/Rechts-Shift Zuweisung inkl. bitweisem AND/XOR/OR	
15	,	Komma	Links→rechts

[14] https://en.cppreference.com/w/c/language/operator_precedence

Table of Contents

Genereller Aufbau

```
[Modifier] Rückgabetyp Funktionsname(Parameter) {
   /* Anweisungsblock mit Anweisungen */
}
```

Beispiel:

```c
#include <stdio.h>

void hilfe(int a) {
  printf("Ich bin die Hilfsfunktion %d\n", a);
}

int main(void) {
  hilfe(1);
  return 0;
}
```

Vorwärtsdeklaration

Funktion muss vor Verwendung bereits deklariert worden sein

```c
#include <stdio.h>

int vorbereitung(void);
void hilfe(int);

int main(void) {
  hilfe(vorbereitung());
  return 0;
}

int vorbereitung() {
  return 1;
}

void hilfe(int a) {
  printf("Ich bin die Hilfsfunktion %d\n", a);
}
```

Alternative: Header

Rekursion[15]

```c
#include <stdio.h>

int divide(int x, int y) {
   if(x >= y)
      return (1 + divide(x - y, y));
   if(x)
      printf("Zahl nicht teilbar -> Rest:
      ↪  %d -> ", x);
   return 0;
  }

int main() {
  printf("%d\n", divide(10, 5));
  printf("%d\n", divide(10, 4));
  return 0;
}
```

```
$ ./recursion
2
Zahl nicht teilbar -> Rest: 2 -> 2
```

int main()	divide(10,5)
return 1+	divide(5,5)
return 1+	divide(0,5)
return 0;	

int main()	divide(10,4)
return 1+	divide(6,4)
return 1+	divide(2,4)
return 0;	

[15] recursion.c

```c
#include <stdio.h>

int divide(int x, int y) {
   if(x >= y)
      return (1 + divide(x - y, y));
   if(x)
      printf("Zahl nicht teilbar -> Rest:
      ↪  %d -> ", x);
   return 0;
  }

int main() {
  printf("%d\n", divide(10, 5));
  printf("%d\n", divide(10, 4));
  return 0;
}
```

```
$ ./recursion
2
Zahl nicht teilbar -> Rest: 2 -> 2
```

int main()	divide(10,5)
return 1+	divide(5,5)
return 1+	divide(0,5)
return 0;	

int main()	divide(10,4)
return 1+	divide(6,4)
return 1+	divide(2,4)
return 0;	

[15] recursion.c

```c
#include <stdio.h>

int divide(int x, int y) {
   if(x >= y)
      return (1 + divide(x - y, y));
   if(x)
      printf("Zahl nicht teilbar -> Rest:
      ↪  %d -> ", x);
   return 0;
  }

int main() {
  printf("%d\n", divide(10, 5));
  printf("%d\n", divide(10, 4));
  return 0;
}
```

```
$ ./recursion
2
Zahl nicht teilbar -> Rest: 2 -> 2
```

`int main()`	`divide(10,5)`
`return 1+`	`divide(5,5)`
`return 1+`	`divide(0,5)`
`return 0;`	

`int main()`	`divide(10,4)`
`return 1+`	`divide(6,4)`
`return 1+`	`divide(2,4)`
`return 0;`	

[15] recursion.c

```c
#include <stdio.h>

int divide(int x, int y) {
   if(x >= y)
      return (1 + divide(x - y, y));
   if(x)
      printf("Zahl nicht teilbar -> Rest:
      ↪  %d -> ", x);
   return 0;
  }

int main() {
  printf("%d\n", divide(10, 5));
  printf("%d\n", divide(10, 4));
  return 0;
}
```

```
$ ./recursion
2
Zahl nicht teilbar -> Rest: 2 -> 2
```

int main()	divide(10,5)
return 1+	divide(5,5)
return 1+	divide(0,5)
return 0;	

int main()	divide(10,4)
return 1+	divide(6,4)
return 1+	divide(2,4)
return 0;	

[15] recursion.c

Signatur [16]:

```c
#include <stdio.h>

int printf(const char * restrict format, ...);
```

Format Specifier [6]: `%[flags][width][.precision][length]specifier`

Beispiel:

```c
#include <stdio.h>

int main (void) {
  short sval = 1;
  double dval = 2.4;

  printf("Der short-Wert ist  %hd\n", sval);
  printf("Der double-Wert ist %f\n", dval);
  return 0;
}
```

[16] Variable Argumentlisten: https://openbook.rheinwerk-verlag.de/c_von_a_bis_z/018_c_stdarg_h_001.htm

Ausgabe Beispiel

```c
#include <stdio.h>

int main()
{
   printf("Char: %c %c\n", 'a', 65);
   printf("Integer: %d\n", 1977);
   printf("Leerzeichen vorweg: %10d\n", 1977);
   printf("Nullen vorweg: %010d\n\n", 1977);

   printf("Zahlensysteme:\n");
   printf("  Dezimal: %d\n", 100);
   printf("  Hexadezimal: %x\n", 100);
   printf("  Oktal: %o\n", 100);
   printf("  Hexadezimal mit Präfix: %#x\n", 100);
   printf("  Oktal mit Präfix: %#o\n\n", 100);

   printf("Gleitkommazahlen:\n");
   printf("  Festkommadarstellung: %4.2f\n", 3.1416);
   printf("  Wiss. Darstellung: %+.0e\n", 3.1416);
   printf("  Wiss. Darstellung: %E\n\n", 3.1416);

   printf("%s\n", "Ein String");
   return 0;
}
```

```
$ ./printf
Char: a A
Integer: 1977 0
Leerzeichen vorweg:       1977
Nullen vorweg: 0000001977

Zahlensysteme:
  Dezimal: 100
  Hexadezimal: 64
  Oktal: 144
  Hexadezimal mit Präfix: 0x64
  Oktal mit Präfix: 0144

Gleitkommazahlen:
  Festkommadarstellung: 3.14
  Wiss. Darstellung: +3e+00
  Wiss. Darstellung: 3.141600E+00

Ein String
```

Eingabemöglichkeiten

- Per Makro beim Kompilieren
  - Ein- und Ausschalten von Features
- (Auslesen von Umgebungsvariablen)
- Übergabe beim Start auf Kommandozeile
- Interaktive Eingabeaufforderung
  - Selten im HPC
- (Einlesen aus Datei)
  - Primäre Vorgehensweise im HPC

Eingabe beim Programmaufruf

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {
   int i;

   for(i=0; i < argc; i++) {
      printf("argv[%d] = %s (%d)\n", i, argv[i], strlen(argv[i]));
   }
   return 0;
}
```

```
$ ./argv_type test 1 23
argv[0] = ./argv_type (11)
argv[1] = test (4)
argv[2] = 1 (1)
argv[3] = 23 (2)
```

Achtung: Alle Eingaben sind Zeichenketten

Eingabe beim Programmaufruf

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {
   int i;

   for(i=0; i < argc; i++) {
      printf("argv[%d] = %s (%d)\n", i, argv[i], strlen(argv[i]));
   }
   return 0;
}
```

```
$ ./argv_type test 1 23
argv[0] = ./argv_type (11)
argv[1] = test (4)
argv[2] = 1 (1)
argv[3] = 23 (2)
```

Achtung: Alle Eingaben sind Zeichenketten

Interaktive Eingaben mit scanf

Signatur:

```c
#include <stdio.h>

int scanf(const char * restrict format, ...);
```

Beispiel:

```c
#include <stdio.h>

int main (void) {
   int i;
   printf("Bitte geben Sie eine Zahl ein : ");
   scanf("%d",&i);      // Wartet auf die Eingabe
   printf("Eingabe: %d\n",i);
   return 0;
}
```

References

[1] Wikipedia: *IEEE 754*.
https://de.wikipedia.org/wiki/IEEE_754

[2] Wikipedia: *Single-precision floating-point format* https://en.wikipedia.org/wiki/Single-precision_floating-point_format

[3] Wikipedia: *Double-precision floating-point format* https://en.wikipedia.org/wiki/Double-precision_floating-point_format

[4] Wikipedia: *Extended precision*
https://en.wikipedia.org/wiki/Extended_precision

[5] Shadertoy: *Floating Point Grid*
https://www.shadertoy.com/view/4tVyDK

[6] cplusplus.com: *printf*
https://cplusplus.com/reference/cstdio/printf/