Syntax und Kontrollstrukturen

Praktikum „C-Programmierung“

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Eugen Betke, Nathanael Hübbe,
Michael Kuhn, (Jakob Lüttgau), Jannek Squar

2019-10-28

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

Declaration vs. Definition

- Deklaration: Bekanntmachung eines Bezeichners
- Zuteilung eines Speicherbereiches an Bezeichner

```
1 // Declaration
2 int max(int a, int b);
3 extern char c;
4
5 // Definition (and Declaration)
6 int max(int a, int b) {  /* ... */  }
7 char c = 'a';
```

▸ Lesestoff

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 Schlüsselwörter

Umfangreiche Syntax unter ▸ C Backus Naur Form[1] oder ▸ Wikipedia

[1] basiert auf Abschnitt A13 aus *The C programming language*, 2nd edition, by Brian W. Kernighan and Dennis M. Ritchie,Prentice Hall, 1988

Literale

Zeichenfolge zur Darstellung von Basistyp-Werten

- Ganzzahlen
  - Dezimal [123]
  - Oktal [0173]
  - Hexadezimal [0x7B],
- Fließkommazahlen
- Suffixe spezifizieren genauen Datentyp
- Zeichenliteral ('A')
- *String* ("Foo"), endet mit `'\0'`

Statements und Expressions

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

Keywords:

```
auto      break     case              char    const          continue
default   do        double            else    enum           extern
float     for       goto              if      inline (C99)   int
long      register  restrict (C99)    return  short          signed
sizeof    static    struct            switch  typedef        union
unsigned  void      volatile          while
```

Neuere Keywords:

```
_Alignas (C11)       _Alignof (C11)      _Atomic (C11)    _Bool (C99)
_Complex (C99)       _Generic (C11)      _Imaginary (C99) _Noreturn (C11)
_Static_assert (C11) _Thread_local (C11)
```

▸ Keywords in C

Keywords:

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

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

// Function-Specific
inline   restrict  return

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

Basisdatentypen

```
// Integer-Typen
char
short
int
long
long long

// Gleitkommazahlen
float
double
long double
```

```
// Integer-Typen (aus limits.h)
unsigned/signed char      2^8 - 1 = 255
unsigned/signed short int 2^16 - 1 = 65535
unsigned/signed int       2^32 - 1 = 4294967295
unsigned/signed long int  2^64 - 1 = 18446744073709551615

// Gleitkommazahlen (IEEE 754)
       sign | exponent (8 bit) | fraction (23 bit)
float  0        00000000            00000000000000000000000
       sign | exponent (11 bit) | fraction (52 bit)
double 0        00000000000         0000000000000000000000000000000000000000000000000000
long double ...

// Qualifiers/Modifier
const char* p;          // read-only (Compilerunterstuetzung)
static unsigned int n;  // Kontext:
                            in File: File-globale Variable
                            in Funktion: Behalte Wert ueber Aufrufe hinweg
```

Fließkommazahlen - Genauigkeit

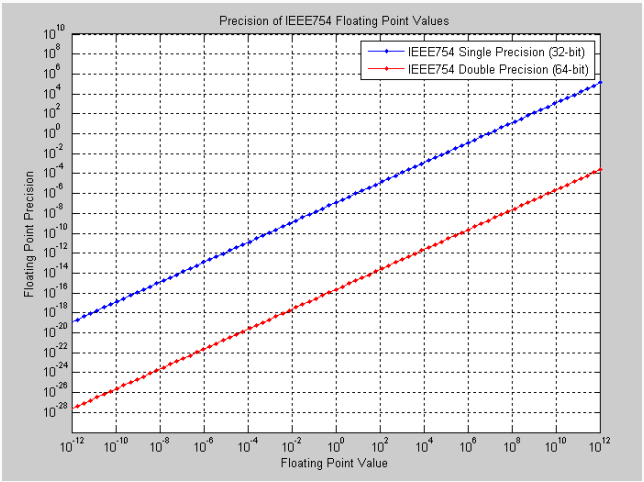


Abbildung 1: By Ghennessey - Own work, CC BY-SA 4.0, ▸ Link

Fließkommazahlen- Genauigkeit

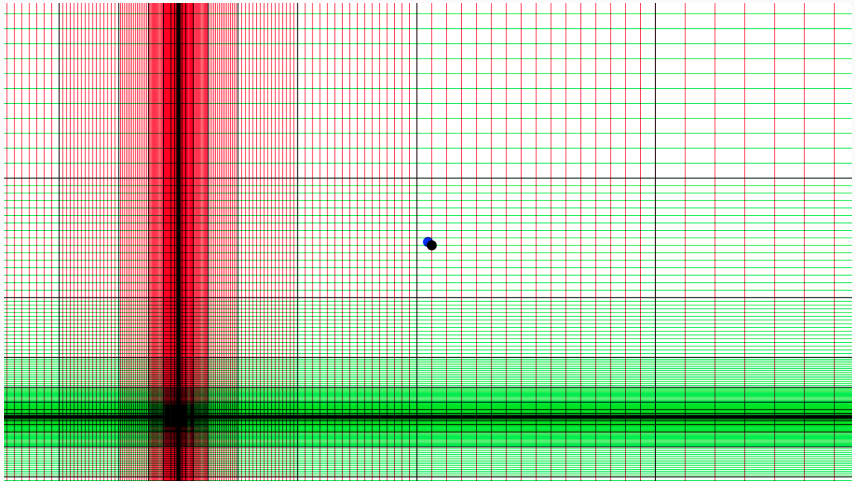


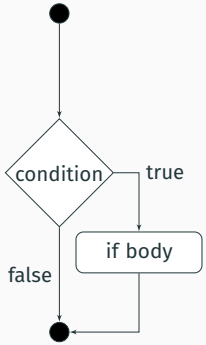
Abbildung 2: Verlust der Genauigkeit ▸ Link

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	Rechts→links
15	,	Komma	Links→rechts

if ...

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

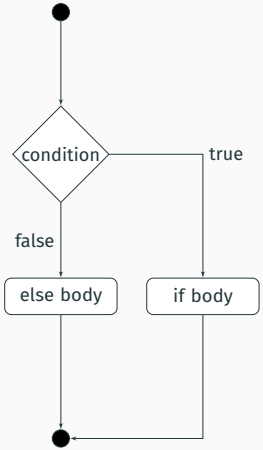
```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int answer = 42;
6
7      if ( answer == 42 )
8      {
9          printf("Here!\n");
10     }
11 }
```



if ... else ...

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

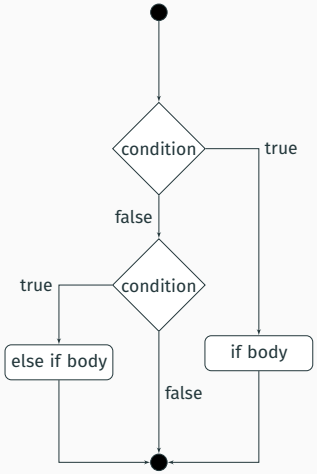
```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int answer = 84;
6      if ( answer == 42 )
7      {
8          printf("Here!\n");
9      }
10     else
11     {
12         printf("Alternative universe!\n");
13     }
14 }
```



if ... else if ...

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

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int answer = 21;
6      if ( answer == 42 )
7      {
8          printf("Here!\n");
9      }
10     else if ( answer == 21 )
11     {
12         printf("Specific alternative universe!\n");
13     }
14 }
```



Integer-Vergleich mit Switch

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

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 }
```

Unäre & binäre Operatoren[3]

Unär

- Negation: `(! condition)`
- Negation: `(not condition)`[2]

Binär

- logisches UND: `(condition && condition)`, `(condition and condition)`
- logisches ODER: `(condition || condition)`, `(condition or condition)`
- ⚠ Verwechslungsgefahr mit `|`, `&` und `=`

Operatoren können beliebig verschachtelt werden

[2] Benötigt `iso646.h`-Header
[3] verwechslung.c

Ternärer Operator

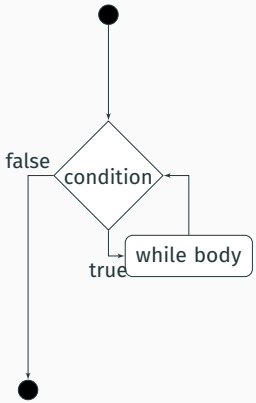
```
1 int a = 5, b = 8;
2 int min;
3
4 // condition ? expression : expression
5 min = (a < b) ? a : b;
```

- praktisch
- verminderte Lesbarkeit

while-Schleife[4]

```
1  while ( condition )
2  {
3      statement;
4      statement;
5  }
```

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int i = 0;
6
7      while ( i < 6 )
8      {
9          printf("%d ", i);
10         i++;
11     }
12 }
```



[4] while-loop.c

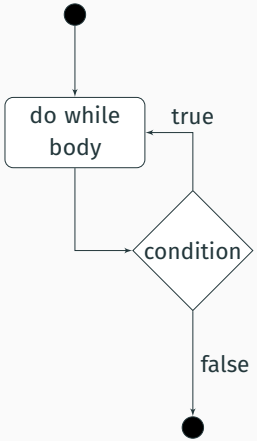
do-while-Schleife[5]

```c
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 );
}
```

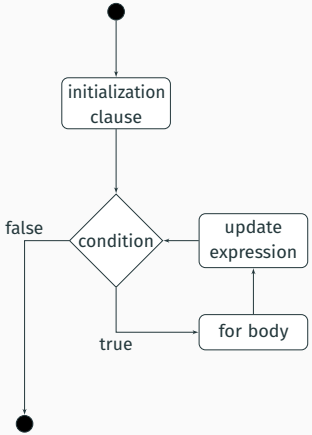


[5] do-while-loop.c

for-Schleife

```
1  for (clause; condition; expression)
2  {
3      statement;
4      statement;
5  }
```

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      for (int i = 0; i < 10; i++)
6      {
7          printf("%d ", i);
8      }
9  }
```



Kontrollierte Sprünge[6]

```
1  for (int i = 0; i < 10; i++)
2  {
3      if ( i == 4 || i == 6 )
4          continue;
5
6      if ( i == 9 )
7          break;
8
9      printf("%d ", i)
10 }
```

[6] for-loop.c

Der Vollständigkeit halber: goto[7]

```
1  goto LABEL;
2  LABEL : Anweisung;
```

```
1  #include <stdio.h>
2
3  int main(void) {
4     int i,j,k;
5     for(i=1; i<10; i++) {
6        for(j=1; j<10; j++) {
7           for(k=1; k<10; k++) {
8              printf("Tiefe Verschachtelungsebene\n");
9              goto RAUS;
10          }
11       }
12    }
13    RAUS : printf("Mit einem Sprung raus hier \n");
14    return 0;
15 }
```

▸ Relevantes XKCD

[7] goto.c

Präprozessor-Tokens

```
1  if       elif     else     endif    defined
2  ifdef    ifndef   define   undef    include
3  line     error    pragma
```

▸ C Präprozessor Keywords

Examples

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

int main(void)
{

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

}
```

Lesestoff

Examples

```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
}
```

▸ Lesestoff

Examples[8]

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

int main(void)
{

#if !defined(DCBA) && (ABCD < 2*4-3)
    printf("3: yes\n");
#endif

}
```

▸ Lesestoff

[8] abcd_complete.c

gcc -D<varname>=<value>[9]

```c
#include <stdio.h>

int main(void)
{
#ifdef VARIANT
    printf("Variant B\n");
#else
    printf("Variant A (default)\n");
#endif
}
```

```c
// gcc program.c && ./a.out
// Result: Variant A (default)

// gcc -DVARIANT program.c && ./a.out
// Result: Variant B
```

[9] DVAR.c