Syntax und Kontrollstrukturen

Praktikum „C-Programmierung“

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

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

2018-10-29

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Statements, Expressions und Literale

```
// Expressions
a + b
a*b / 14
a >= b

// Statements
;
control_statement {
    statement;
    statement;
}
if ( expression ) { statement; statement; } else statement;

// Literale (vgl. Datentypen):                  String-Literal:
42    1.42    0x8483    'c'                     "abc"
```

Declaration vs. Definition

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

// Definition (and Declaration)
int max(int a, int b) {  /* ... */  }
char c = 'a';
```

Ist die Allokation von Speicher für Variablen oder Programmcode involviert handelt es sich um eine Definition.

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)
```

Compilerabhängige Keywords von Extensions:

```
asm     fortran
```

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
```

Datentypen und Qualifier

```
char
double
float
int
long
short
```

Datentypen und Qualifier

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

// Gleitkommazahlen
float
double
long double
```

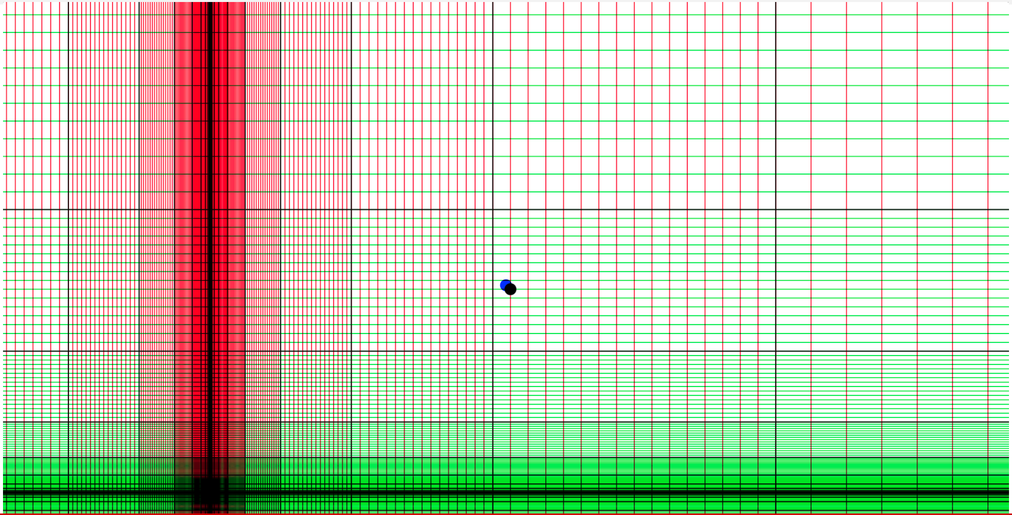
Datentypen und Qualifier

```
// Integer-Typen (also consult #include <limits.h>)
unsigned/signed char       2^8 - 1 = 255                           -128 to 127
unsigned/signed short     2^16 - 1 = 65535                     -32768 to 32767
unsigned/signed int       2^32 - 1 = 4294967295                              :
unsigned/signed long      2^64 - 1 = 18446744073709551615

// Gleitkommazahlen    z.B. nach  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 (help the compiler help you)
static unsigned int n;  // context dependent:
                           in file: a file global variable
                           in function: retain value across invocations
```

Floating Point Loss of Precision



Operatoren

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(type){list}	Compound literal(C99)	
2	++ --	Prefix increment and decrement	Right-to-left
	+ -	Unary plus and minus	
	! ∼	Logical NOT and bitwise NOT	
	(type)	Type cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of[note 1]	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and <= respectively	
	>>=	For relational operators > and >= respectively	
7	== !=	For relational = and != respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
(13)	?:	Ternary conditional (parsed as if parenthesized)	Right-to-Left
14	=	Simple assignment	Right-to-Left
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

Quelle: `https://en.cppreference.com/w/c/language/operator_precedence`

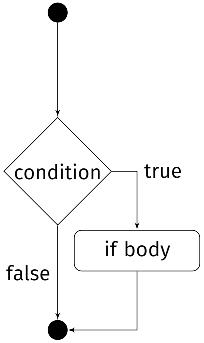
if, else, else if

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

```
#include <stdio.h>

int main(void)
{
    int answer = 42;

    if ( answer == 42 ) {
        printf("Here!\n");
    }
}
```



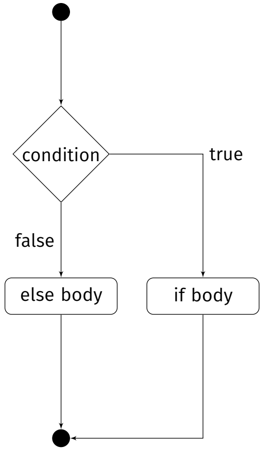
if, else, else if

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

```c
#include <stdio.h>

int main(void)
{
    int answer = 84;

    if ( answer == 42 ) {
        printf("Here!\n");
    } else {
        printf("Alternative universe!\n");
    }
}
```



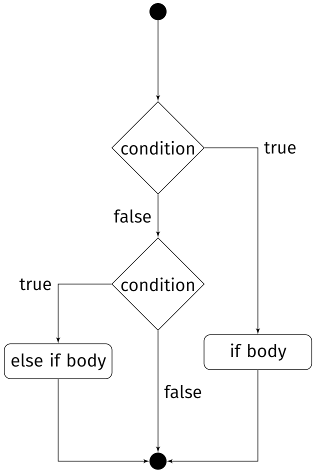
if, else, else if

```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");
    }
}
```



Switch

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



}
```

Switch

```
switch (expression) {
    case A:
    case B:
        statement;
        break;
    case C:
        statement;
                       /*FALLTHROUGH*/
    case D:
        statement;
        break;


}
```

Switch: Standardfall

```
switch (expression) {
    case A:
    case B:
        statement;
        break;
    case C:
        statement;
                       /*FALLTHROUGH*/
    case D:
        statement;
        break;
    default:
        statement;
}
```

Bedingte Expression / ternärer Operator

```c
int a = 5, b = 8;
int min;

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

- Manchmal praktisch (etwa mit return) vermindert aber häufig die Lesbarkeit

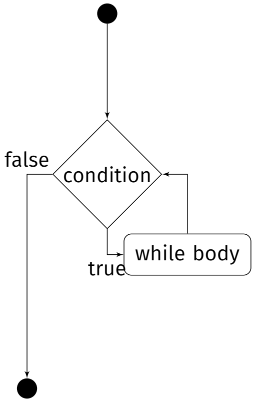
while, do while

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

```
#include <stdio.h>

int main(void)
{
    int i = 0;

    while ( i < 6 ) {
        printf("%d ", i);
        i++;
    };
    // Result: 0 1 2 3 4 5
}
```



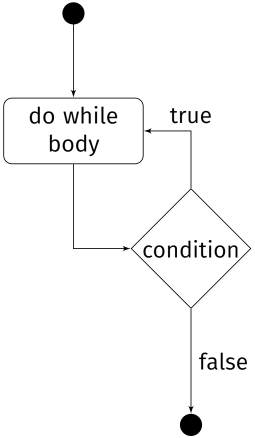
while, do while

```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 );
    // Result: 0
}
```

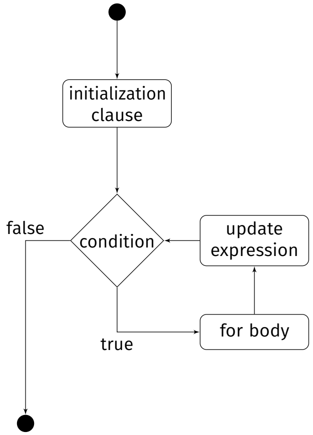


for

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

```
#include <stdio.h>

int main(void)
{
    for (int i = 0; i < 10; i++) {
        printf("%d ", i);
    }
}
// Result: 0 1 2 3 4 5 6 7 8 9
```



continue und break

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

    if ( i == 9 )
        break;

    printf("%d ", i)
}

// Result: 0 1 2 3 5 8
```

Präprozessor-Tokens

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

Siehe auch: `https://en.cppreference.com/w/c/keyword`

Examples

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

int main(void)
{

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

}
```

Siehe auch: `https://en.cppreference.com/w/c/preprocessor/conditional`

Examples

```
#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
}
```

Siehe auch: `https://en.cppreference.com/w/c/preprocessor/conditional`

Examples

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

int main(void)
{

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



}
```

Siehe auch: `https://en.cppreference.com/w/c/preprocessor/conditional`

gcc -D<varname>=<value>

```c
#include <stdio.h>

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

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

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