



Ministerium für Kultus, Jugend  
und Sport Baden-Württemberg  
  
Koordinierungsstelle für Abschluss-  
prüfungen Berufsschule-Wirtschaft

Landesfachausschuss für die Ausbildungsberufe  
• Elektroniker/ -in für Geräte und Systeme (FA 205/1)  
• Systemelektroniker/ -in (FA 205/2)

# FA 205

## Technische Richtlinien für Unterricht und Prüfung

**Festlegung wichtiger Begriffe  
Stand**

**04. März 2016**

### **1 Übersicht der Programmiersprache C**

- 1.1 Grundstruktur eines C-Programms
- 1.2 Verwendete Datentypen
- 1.3 Zahlensysteme
- 1.4 Verwendete Operatoren
- 1.5 Kontrollstrukturen
  - 1.5.1 Verzweigungen
  - 1.5.2 Fallauswahl
  - 1.5.3 Schleifen
    - 1.5.3.1 for-Schleife (zählergesteuert)
    - 1.5.3.2 while-Schleife (kopfgesteuert)
    - 1.5.3.3 do-while-Schleife (fußgesteuert)
- 1.6 Funktionen
  - 1.6.1 Deklaration von Funktionen
  - 1.6.2 Definition von Funktionen
  - 1.6.3 Funktionsaufruf

### **2 Blockschaltbild des Mikrocontrollers**

### **3 Bibliotheken**

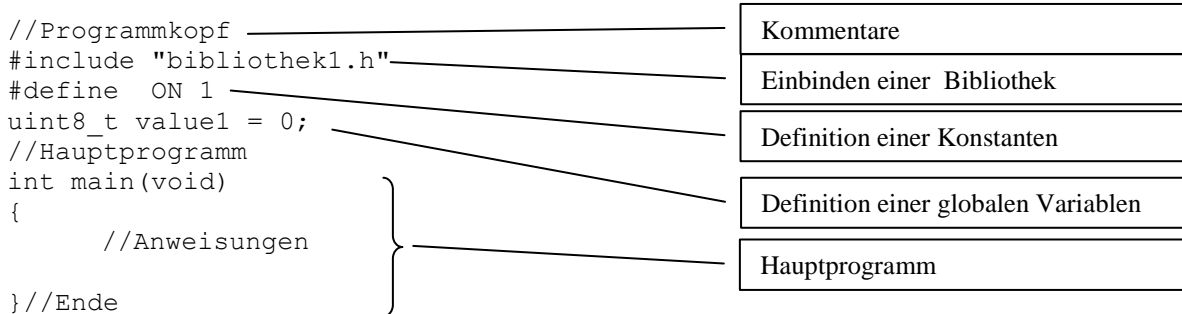
- |     |             |                 |                 |                |
|-----|-------------|-----------------|-----------------|----------------|
| 3.1 | Bibliothek: | delay.c         | delay.h         | } controller.h |
| 3.2 | Bibliothek: | in_out.c        | in_out.h        |                |
| 3.3 | Bibliothek: | interrupt.c     | interrupt.h     |                |
| 3.4 | Bibliothek: | lcd.c           | lcd.h           |                |
| 3.5 | Bibliothek: | communication.c | communication.h |                |

#### **Hinweise zur Anwendung der Technischen Richtlinien:**

- Die Technischen Richtlinien sind im Unterricht als Standards für die landeseinheitliche Abschlussprüfung einzusetzen.
- Die verwendeten Begriffe und Darstellungen stellen prinzipielle Regelungen dar. In den Aufgaben der Abschlussprüfungen können im Einzelfall Abweichungen auftreten.
- Die Technischen Richtlinien ergänzen das an der Schule eingeführte Tabellenbuch.

# 1 Übersicht der Programmiersprache C

## 1.1 Grundstruktur eines C-Programms



## 1.2 Verwendete Datentypen

ANSI C Bezeichnung	Typdefinitionen	Größe	Wertebereich
signed char	int8_t	1 Byte	-128...+127
unsigned char	uint8_t	1 Byte	0...255
signed int	int16_t	2 Byte	-32768...+32767
unsigned int	uint16_t	2 Byte	0...65535
signed long	int32_t	4 Byte	-2147483648...2147483647
unsigned long	uint32_t	4 Byte	0...4294967295

Syntax:

//Variablendeklaration

<Datentyp> <Variablenname>;

Beispiel:

uint8\_t value1;

## 1.3 Zahlensysteme

Zahlensystem	Kennzeichen	Beispiel
Dezimal		wert = 12;
Hexadezimal	0x	wert = 0xC;

## 1.4 Verwendete Operatoren

Mathematische Operatoren		
Operation	Beschreibung	Beispiel
+	Addition	value1 = 34 + 12
-	Subtraktion	value1 = 34 - 12;
*	Multiplikation	value1 = value1 * 2;
/	Division	value1 = 50 / 7; → 7
%	Modulo Operation. Liefert den Rest einer Ganzzahl-Division	value1 = 50 % 7; → 1
++	Inkrementieren, +1 dazurechnen	value1++;
--	Dekrementieren, -1 abziehen	value1 --;
Bitweise Operatoren		Beispiel: uint8_t value1 = 0xF0;
&	UND	value2 = value1 & 0x81; → 0x80
	ODER	value2 = value1   0x81; → 0xF1
^	XOR	value2 = value1 ^ 0x81; → 0x71
~	Invertierung	value2 = ~value1; → 0x0F
<<	Nach links schieben	value2 = value1 << 1; → 0xE0
>>	Nach rechts schieben	value2 = value1 >> 3; → 0x1E
Logische Operatoren		
==	Gleich	if (value1 == 10)
!=	Ungleich	while (Taster != 0)
>	Größer	while (value1 > 100)
<	Kleiner	while (value1 < 100)
>=	Größer gleich	while (value1 >= 100)
<=	Kleiner gleich	while (value1 <= 100)
!	NICHT	if (!Taster)
&&	UND	while ((value1 > 10) && (value1 < 100))
	ODER	while ((value1 < 10)    (value2 > 100))

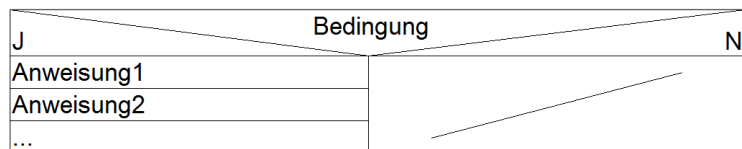
## 1.5 Kontrollstrukturen

### 1.5.1 Verzweigungen

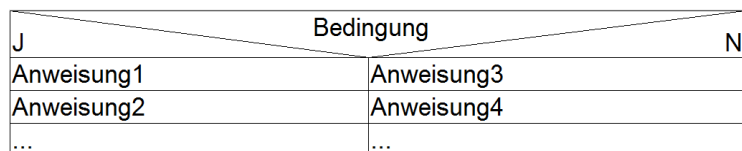
Syntax:

```
if (<Bedingung>)
{
    <Anweisung1>;
    <Anweisung2>;
    ...
}
```

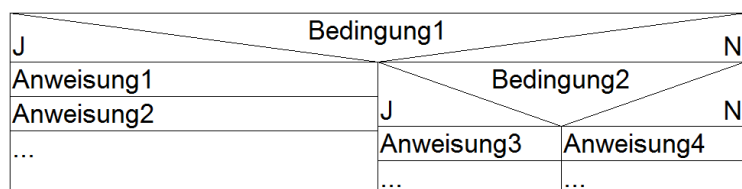
Struktogramm:



```
if (<Bedingung>)
{
    <Anweisung1>;
    <Anweisung2>;
    ...
}
else
{
    <Anweisung3>;
    <Anweisung4>;
    ...
}
```



```
if (<Bedingung1>)
{
    <Anweisung1>;
    <Anweisung2>;
    ...
}
else if (<Bedingung2>)
{
    <Anweisung3>;
    ...
}
else
{
    <Anweisung4>;
    ...
}
```



## Beispiele:

```
if (value1 < 100)
{
    value1++;
}
```

```
if (value1 < 100)
{
    value1++;
}
else
{
    value1 = value2;
}
```

```
if (value1 < 100)
{
    value1++;
}
else if (value1 < 200)
{
    value1 = value2;
}
else
{
    value1 = value3;
}
```

## 1.5.2 Fallauswahl

### Syntax:

```
switch (<Vergleichswert>)
{
    case <Wert1>:
        <Anw.1>;
        <Anw.2>;
        ...
        break;
    case <Wert2>:
        <Anw.3>;
        <Anw.4>;
        ...
        break;
    ...
    default:
        <Anw.5>;
        <Anw.6>;
        ...
        break;
}
```

### Struktogramm:

Vergleichswert		
Wert1	Wert2	default
Anw.1	Anw.3	Anw.5
Anw.2	Anw.4	Anw.6
...	...	...

### Beispiel:

```
switch (value1)
{
    case 1:
        value1++;
        break;
    case 2:
        value1 = value2;
        break;
    default:
        value2 = value1;
        break;
}
```

## 1.5.3 Schleifen

### 1.5.3.1 for-Schleife (zählergesteuert)

#### Syntax:

```
for (<Startwert>; <Bedingung>; <Schrittweite>)
{
    <Anweisung1>;
    <Anweisung2>;
    ...
}
```

#### Beispiel:

```
int8_t i;
for (i = 0; i < 100; i++)
{
    value1++;
}
```

### Struktogramm:

Für Startwert, Bedingung, Schrittweite	
Anweisung1	
Anweisung2	
...	

### 1.5.3.2 while-Schleife (kopfgesteuert)

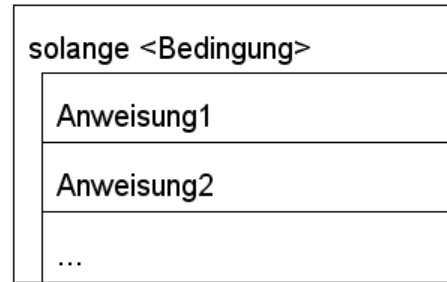
Syntax:

```
while (<Bedingung>)  
{  
    <Anweisung1>;  
    <Anweisung2>;  
    ...  
}
```

Beispiel:

```
value1 = 0;  
while (value1 < 100)  
{  
    value1++;  
}
```

Struktogramm:



### 1.5.3.3 do-while-Schleife (fußgesteuert)

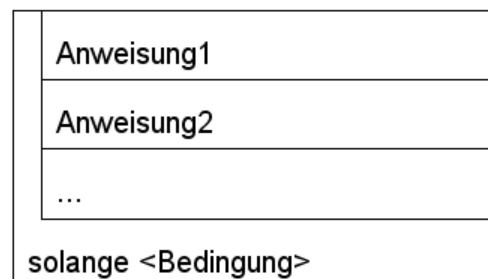
Syntax:

```
do  
{  
    <Anweisung1>;  
    <Anweisung2>;  
    ...  
} while (<Bedingung>;
```

Beispiel:

```
value1 = 0;  
do  
{  
    value1++;  
} while (value1 < 100);
```

Struktogramm:



## 1.6 Funktionen

### 1.6.1 Deklaration von Funktionen

Syntax:

```
<Datentyp Rückgabewert> funktionsname(<Datentyp> Parameter1, ...);
```

Beispiele:

```
void pwm_init(void);           // ohne Rückgabewert, ohne Parameter  
void lcd_byte(uint8_t value1); // ohne Rückgabewert, mit einem Parameter  
uint8_t adc_in1(void);         // mit Rückgabewert, ohne Parameter
```

### 1.6.2 Definition von Funktionen

Syntax:

```
<Datentyp Rückgabewert> funktionsname(<Datentyp> Parameter1, ...)  
{  
    //Anweisungen  
    return <Rückgabewert>;  
}
```

Beispiel:

```
uint16_t addieren(uint8_t z1, uint8_t z2)  
{  
    uint16_t ergebnis;  
    ergebnis = z1 + z2;  
    return ergebnis;  
}
```

### 1.6.3 Funktionsaufruf

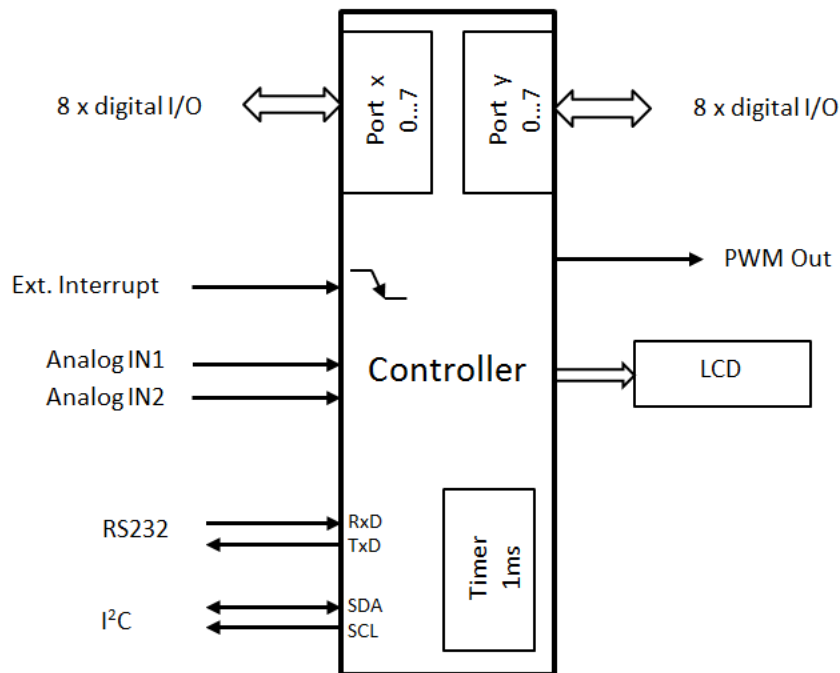
Syntax:

```
wert = funktionsname(Parameter1, ...);
```

Beispiel:

```
uint16_t value3 = 0;
int main(void)
{
    ...
    value3 = addieren(5, 123);    // Funktionsaufruf mit Rückgabewert
    ...
}
```

## 2 Blockschaltbild des Mikrocontrollers



## 3 Bibliotheken

Alle Header-Dateien der Technischen Richtlinie werden in der Datei „**controller.h**“ in ein richtlinienkonformes Projekt eingebunden.

### 3.1 Bibliothek: delay.c delay.h

#### Delay

Verzögert den Programmablauf für die angegebene Zeitdauer.

```
delay_ms ( uint16_t value);           // value = 0...65535 (x 1ms)
delay_100us ( uint16_t value );       // value = 0...65535 (x 100µs)
```

## 3.2 Bibliothek: in\_out.c

## in\_out.h

### Bit Ein-/Ausgabe

```
bit_init ( uint8_t port, uint8_t bitnr, uint8_t direction );
bit_write ( uint8_t port, uint8_t bitnr, uint8_t value );
uint8_t value = bit_read ( uint8_t port, uint8_t bitnr );
```

Parameterliste:

**port:** PORTx, PORTy  
(8051: PORT0, PORT1,...  
AVR: \_PORTB\_, \_PORTC\_, ...)  
**bitnr:** 0...7  
**direction:** IN = 0, OUT = 1  
**value:** 0/1

### Byte Ein-/Ausgabe

```
byte_init ( uint8_t port, uint8_t direction );
byte_write ( uint8_t port, uint8_t value );
uint8_t value = byte_read ( uint8_t port );
```

Parameterliste:

**value:** 0 ... 255

### Pulsweitenmodulation

Ein digitaler Ausgang (PWM Out) zur Ausgabe eines pulswidenmodulierten Signals.

```
pwm_init ( ); // Initialwert für Tastgrad 50%
pwm_start ( );
pwm_stop ( );
pwm_duty_cycle ( uint8_t value );
```

Parameterliste:

**value:** 0 ... 255 (entspricht Tastgrad 0...100%)  
Initialwert: 127

### Analog-Digital Konverter (ADC)

Zwei analoge Eingänge, je 8 Bit Auflösung.

```
adc_init ( );
uint8_t value = adc_in1 ( );
uint8_t value = adc_in2 ( );
```

Rückgabewert:

**value:** 0 ... 255

### Bsp.: Bit-Ein-Ausgabe

```
#include "controller.h"
uint8_t temp;

int main( void )
{
    bit_init( PORTx, 2, IN ); // Taster
    bit_init( PORTy, 0, OUT ); // LED

    while (1)
    {
        temp = bit_read( PORTx, 2 );
        bit_write( PORTy, 0, temp );
    }
}
```

### Bsp.: PWM

```
#include "controller.h"

int main( void )
{
    pwm_init( );

    pwm_duty_cycle( 63 ); // Tastgrad = 25%
    pwm_start( );
    delay_ms( 5000 );
    pwm_stop( );

    while(1);
}
```

### Bsp.: ADC

```
#include "controller.h"
uint8_t wert;

int main( void )
{
    adc_init( );
    byte_init( PORTx, OUT );

    while(1)
    {
        wert = adc_in1( ); // ADC einlesen
        byte_write( PORTx, wert );
    }
}
```

### 3.3 Bibliothek: interrupt.c

#### Externer Interrupt

Bei fallender Flanke am Interrupteingang wird die Interruptservice-Routine `ext_interrupt_isr()` aufgerufen.

```
ext_interrupt_init ( ext_interrupt_isr );
ext_interrupt_enable ( );
ext_interrupt_disable ( );
ext_interrupt_isr ( );
```

#### Timer (Quelle ist Systemtakt)

Interner Timer ruft alle 1ms die `timer1ms_isr()` auf.

```
timer1ms_init ( timer1ms_isr );
timer1ms_enable ( );
timer1ms_disable ( );
timer1ms_isr ( );
```

Hinweis: Beide Interrupts haben gleiche Priorität!

### interrupt.h

#### Bsp.: externer Interrupt

```
#include "controller.h"
uint8_t wert = 0;

int main( void )
{
    ext_interrupt_init( ext_interrupt_isr );
    byte_init( PORTx, OUT );

    ext_interrupt_enable( );
    while(1);    // Endlos
}

void ext_interrupt_isr( void )
{
    byte_write( PORTx, wert++ );
}
```

### 3.4 Bibliothek: lcd.c

#### LC-Display

Funktionen zur Ansteuerung eines Textdisplays.

```
lcd_init ( );
lcd_clear ( );
lcd_setcursor ( uint8_t row, uint8_t column );
lcd_print ( uint8_t text[ ] );           // Bsp.: "Hallo Welt!"
lcd_char ( uint8_t ascii );              // 'A', 'B', ':', '?', ....
lcd_byte ( uint8_t byte );               // 0 ... 255
lcd_int ( uint16_t word );                // 0 ... 65535
```

Parameterliste:

**row:** 1, 2, ... (Zeile)  
**column:** 1, 2, ... (Spalte)  
**text [ ]:** Zeichenkette ('\0'-terminiert)  
**ascii:** Einzelnes Zeichen auf dem Display darstellen.  
**byte:** 8-Bit Zahl wird als 3-stellige Zahl (ohne führende Nullen) auf Display dargestellt.  
**word:** 16-Bit Zahl (5-stellige Darstellung)

### lcd.h

#### Bsp.: LCD

```
#include "controller.h"
uint8_t meinText[ ] = "Hallo Welt!";

int main( void )
{
    lcd_init();
    lcd_setcursor( 1,1 );
    lcd_print( meinText );
    lcd_setcursor( 2,1 );
    lcd_byte( 155 );
    lcd_char ( 'V' );

    while(1);    // Endlos
}
```



## 3.5 Bibliothek: communication.c communication.h

### RS232

Funktionen zur seriellen Kommunikation.

```
rs232_init ( );  
    // 9600 Baud, 1 Startbit, 8 Datenbit, 1 Stoppbit  
uint8_t value = rs232_get ( );  
    // Liefert \0, wenn kein Zeichen empfangen  
rs232_put ( uint8_t value );  
    // Ausgabe eines Bytes  
rs232_print ( uint8_t text[ ] );  
    // Ausgabe eines Strings
```

Parameterliste:

**value:** Byte (8 Bit)  
**text [ ]:** Zeichenkette ('\0'-terminiert)

#### Bsp.: RS232

```
#include "controller.h"  
uint8_t meinText[ ] = "Controller sind toll!";  
  
int main( void )  
{  
    rs232_init( );  
  
    rs232_print( meinText );  
    rs232_print( "...aber klar!\n" );  
  
    // Warten bis Zeichen != '\0'  
    while (rs232_get( ) == '\0');  
    // Ein Zeichen senden  
    rs232_put( 'A' );  
  
    while(1);    // Endlos  
}
```

### I2C-Bussystem

Funktionen zur Kommunikation mit I2C-Bus Komponenten.

```
i2c_init ( );  
i2c_start ( );  
i2c_stop ( );  
uint8_t ack = i2c_write ( uint8_t value );  
uint8_t value = i2c_read ( uint8_t ack );
```

Parameterliste:

**ack:** ACK = 0, NACK = 1  
**value:** Byte (8 Bit)

#### Bsp.: I2C

```
#include "controller.h"  
#define ADDR_R 0x41  
#define ADDR_W 0x40  
  
uint8_t wert;  
  
int main( void )  
{  
    i2c_init( );  
    i2c_start( );  
    i2c_write( ADDR_R );  
    wert = i2c_read( NACK );  
    i2c_stop( );  
  
    while(1);    // Endlos  
}
```