

In den nachfolgenden Aufgaben wird, wenn nicht explizit anders angegeben, vorausgesetzt, dass die Programmierung für ein System mit einer 32 Bit ARM CPU erfolgt.

Achten Sie immer auf die Einhaltung der „Coding Rules“!

Aufgabe 1: Call-by-value, call-by-reference

Mit der Struktur *Vorlesung* sollen die Vorlesungen verwaltet werden. Dazu sollen zwei Funktionen entwickelt werden: *setTitel* und *printTitel*.

1. Ergänzen Sie die Funktionsdefinition von *setTitel* um die Parameter *vorlesung* und *titel*. *vorlesung* soll per **call-by-reference** übergeben werden.
2. Ergänzen Sie den Rumpf von *setTitel*.
3. Ergänzen Sie die Funktionsdefinition von *printTitel* um den Parameter *vorlesung*. *vorlesung* soll per **call-by-value** übergeben werden.
4. Ergänzen Sie den Rumpf von *printTitel*.
5. Ergänzen Sie *main* um Aufrufe von *setTitel* und *printTitel*.
6. Welche Vorteile hat call-by-reference gegenüber call-by-value?

main.c

```
struct Vorlesung {
    char titel[30];
    int semester;
}vorlesung;

void setTitel(_____) {
    strncpy(_____) ;
}

void printTitel(_____) {
    printf( "Titel=%s", _____) ;
}

int main(void){
    char*titel = "C Programmieren";

    return 0;
}
```

Aufgabe 2: Programmiersprache C

1. Geben Sie an, wie viele Bytes die nachfolgenden Definitionen in einem 32-Bit System benötigen:

```
int feld1[]={1,2,3,4};
```

Anzahl Bytes:

```
struct Test{ char feld2[8]; int n; } test[] =  
            { {"12", 1}, {"123", 2}, {"1", 3}};
```

Anzahl Bytes:

2. Nachfolgende Anweisungen befinden sich außerhalb von Funktionen. Nennen Sie den Unterschied zwischen den beiden Anweisungen. Benutzen Sie auch Fachausdrücke.

(a) `double x;`

(b) `extern double y;`

3. Erstellen Sie ein allgemeines, universell einsetzbares Makro `MULT(a, b)`, das das Produkt von `a` und `b` berechnet.

4. Folgendes Unterprogramm wurde erstellt. Parameter `in` und der Rückgabewert sollen gültige C-Strings sein. Welche groben Fehler sind dem Programmierer unterlaufen?

```
char* reverse( char* in, int inlen){  
    char out[inlen];  
    int i;  
    for(i=0;i<inlen;i++){  
        out[inlen-i]=in[i];  
    }  
    return out;  
}
```

Aufgabe 3: Zeiger

Nebstehend sind einige globale Variablen definiert.
Definieren Sie die folgenden Zeiger und weisen ihnen den geforderten Wert zu:

```
struct Spieler {  
    char* name;  
    int  tore;  
    char  land[20];  
    struct Spieler*mannschaft[11];  
} spieler;  
  
struct Spieler bestenliste[10];
```

1. Zeiger auf *spieler*:
2. Zeiger auf *bestenliste*:
3. Zeiger auf *bestenliste[3]*:
4. Zeiger auf den Namen des Spielers *bestenliste[6]*:
5. Zeiger auf *tore* von *spieler*:
6. Zeiger auf *name* von *spieler*:
7. Zeiger auf *land* von *spieler*:
8. Zeiger auf *mannschaft[3]* von *spieler*:

}

Aufgabe 5: Strings

1. Erstellen Sie ein allgemeines Unterprogramm *copystring*, welches die Zeichen eines Quellstrings in einen Zielspeicher kopiert. Der Zielspeicher soll vom aufrufenden Programm bereitgestellt werden. Dabei ist es wichtig, dass kein Pufferüberlauf stattfinden kann. Reicht der Platz im Zielspeicher nicht aus, sollen so viele Zeichen wie möglich kopiert werden. Im Zielspeicher soll aber nach Abarbeitung der Funktion auf jeden Fall ein gültiger C-String stehen. Rückgabewert der Funktion soll die Länge des gesamten Quellstrings sein, auch wenn der Speicherplatz nicht ausgereicht hat. Es dürfen keine Bibliotheksfunktionen verwendet werden. Überlegen Sie sich als erstes, welche Parameter die Funktion benötigt.
2. Erstellen Sie die *main*-Funktion und zeigen Sie beispielhaft die Verwendung der von Ihnen erstellten Funktion. Definieren Sie den Ziel- und Quellstring. Der Quellstring soll den Text „HAW“ enthalten. Es soll überprüft werden, ob die Funktion erfolgreich ausgeführt worden ist.

Aufgabe 6: Strukturen und dynamische Speicherverwaltung

Es soll eine verkettete Liste für die Verwaltung der Fußballnationalmannschaft erstellt werden.

1. Erstellen Sie die Funktion ***addToList***, die einen Spieler in die Liste hinter das Element ***spieler*** einfügt. Die Funktion soll so aufgebaut sein, dass die Aufrufe, die in ***erstellen*** gezeigt sind, eine Liste in der Reihenfolge [Startelement, „Gomez“, „Klose“] erzeugen. Beachten Sie dabei, dass die lokalen Variablen nach dem Aufruf von ***erstellen*** nicht mehr gültig sind!
2. Ergänzen Sie die for-Anweisung in der Funktion main so, dass alle Spieler der Liste ausgegeben werden.

```
typedef struct Spieler{
    char          *name;
    int           nr;
    struct Spieler *next;
} Spieler;

//Startelement der Liste
Spieler spielerListe = {.name = "", .nr=0, .next=NULL};
```

```
Spieler *add( Spieler* spieler, char* name, int nr ){
```

```
}

int erstellen(void){
    char name1[] = "Neuer";
    int nr1 =1;
    char name2[] = "Weidenfeller";
    int nr2 =22;
    Spieler *spieler = &spielerListe;
    spieler = add( spieler, name1, nr1 );
    if(spieler==0)return -1;
    spieler = add( spieler, name2, nr2 );
    if(spieler==0)return -1;
    return 0
}

int main( void ){
    Spieler* s;
    erstellen();

    for(                  ){
        printf( "%s\n", s->name );
    }
    return 0;
}
```

Aufgabe 7: Programmiersprache C

1. Geben Sie an, wie viele Bytes die nachfolgenden Definitionen in einem 32-Bit System benötigen:

```
char* feld1[]={ "12", "123", "1"};  
Anzahl Bytes:
```

```
char feld2[][4]={ "12", "123", "1"};  
Anzahl Bytes:
```

2. Innerhalb einer Funktion steht die Anweisung:
`int x = 5;`
In welchem Speicherbereich wird diese Variable angelegt?
3. In einem Modul außerhalb von Funktionen steht die Anweisung:
`int y;`
Handelt es sich um eine Definition oder eine Deklaration? Begründung!

4. Geben Sie den Code an, der vom Präprozessor bei Verwendung der nachfolgenden Anweisungen erzeugt wird:
`#define INCREMENT(x,limit) x < limit ? x++ : x`
`int x; int z=5; int max=3;`
`...`
`x = INCREMENT(z,max+2);`

Aufgabe 8: Datentypen

Unten stehende Definitionen von *titel* und *data* sind vorgegeben. Tragen Sie in die Tabelle die Datentypen ein, die sich aus den jeweiligen Ausdrücken ergeben.

```
char* titel = "Master";
```

```
int data[] = {5,9};
```

Ausdruck	resultierender Datentyp
titel	
&titel	
titel[3]	
&titel[3]	
*titel	
data	
*data	
&data	

Aufgabe 9: Speicherbelegung (10 Punkte)

Gegeben sind die globalen Variablen für eine 32-Bit ARM7-CPU :

```
short nummern[][3] = {{1,2},{6,7,8}};
```

```
short *pnummern = nummern[1];
```

```
char name[]="123";
```

- Tragen Sie in die nebenstehende Tabelle die Speicherbelegung ein. Nehmen Sie an, dass der Compiler die Variablen beginnend mit *nummern* der Reihe nach ohne Lücken im Speicher ab Adresse 0x1000 aufsteigend ablegt.

	Speicher (Bytes)
0x1000	
0x1001	
0x1002	
0x1003	
0x1004	
0x1005	
0x1006	
0x1007	
0x1008	
0x1009	
0x100A	
0x100B	
0x100C	
0x100D	
0x100E	
0x100F	
0x1010	
0x1011	
0x1012	
0x1013	
0x1014	
0x1015	
0x1016	
0x1017	

Aufgabe 10: Module

Das Modul *studium* soll eine globale Funktion *setvorlesung* exportieren. Die Struktur *Vorlesung* ist wie folgt definiert:

```
struct Vorlesung{
    char  titel[30];
    int   semester;
}
```

Das *main*-Modul soll diese Funktion benutzen.

1. Ergänzen Sie *setvorlesung* so, dass *title* in *v* eingetragen wird.
2. Ergänzen Sie *main* um den Aufruf von *setvorlesung*, so dass der angegebene Titel in *vorlesung* eingetragen wird.
3. Ergänzen Sie die dargestellten Module um alle notwendigen Anweisungen, so dass ein sicherer Export der Funktion *setvorlesung* aus dem Modul *studium* erfolgt.
4. Kennzeichnen Sie die Definition von *setvorlesung*.
5. Kennzeichnen Sie die Deklaration von *setvorlesung*.

studium.c

```
void setvorlesung(
    struct Vorlesung* v, char* title ){

}
```

studium.h

main.c

```
struct Vorlesung vorlesung;

int main(void){
    char*title = "C Programmieren";

    return 0;
}
```

Aufgabe 11: C-Syntax

Das nachfolgende Programm soll die Anzahl der Buchstaben 'i', die in einer Textdatei enthalten sind, ermitteln. Leider haben sich eine Reihe von Fehlern eingeschlichen, die das fehlerfreie Compilieren und eine richtige Ausgabe verhindern. Beseitigen Sie die Fehler, so dass das Programm nicht nur ohne Fehler und Warnungen übersetzt werden kann und eine richtige Ausgabe produziert, sondern auch den wesentlichen Vorgaben der üblichen Programmierpraxis bzw. „Coding Rules“ entspricht.

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

#define FILENAME test.dat
#define MODE      rb
#define EQUAL( a, b ) a==b
FILE* f;

int main(void){
    int cnt = 0;

    char* zeichenfolge[] = malloc( 5000 );

    f = fopen( FILENAME, MODE );

    fread( zeichenfolge, 1, 5000, f );

    fclose( f );

    for( char* p = zeichenfolge; *p != 0; p++ ){

        cnt += EQUAL( *p, "i" );

    }

    printf("Anzahl i: %d\n", cnt );

}
```

Aufgabe 12: Bitmanipulationen

Aus einem am Mikrocontroller angeschlossenen Gerät wird ein 6-Bit vorzeichenbehafteter Messwert ausgelesen und wie unten dargestellt in einer 8-Bit Variablen temporär als Rohwert gespeichert. Dabei wurden die Bits 6 und 7 zunächst auf 0 gesetzt.

1. Tragen Sie die entsprechende Darstellung als **vorzeichenrichtige 8-Bit Binärzahl** in die Tabelle ein.

Rohwert: 6-Bit Binärwert								Dezimalwert		8-Bit Binärwert							
5				0						7				0			
0	0	0	0	0	0	0	0	0									
0	0	0	0	0	1	0	0	4									
0	0	1	1	1	1	1	1	-1									
0	0	1	1	1	1	0	0	-4									
Maximaler Wert:																	
0	0																
Minimaler Wert:																	
0	0																

2. Ergänzen Sie die Tabelle um den maximal und minimal möglichen Wert .
3. Welche Art der vorzeichenbehafteten Zahlendarstellung wird hier verwendet?
4. Erstellen Sie ein Unterprogramm in C, dass die Konvertierung von 6-Bit nach 8-Bit durchführt. Dieses Programm bekommt den 6-Bit Messwert in dem Parameter **rohwert** wie in der linken Tabelle dargestellt übergeben und soll den auf 8-Bit gewandelten Wert zurückgeben.

```
int8_t convert( int8_t rohwert ){
```

```
}
```

Aufgabe 13: Strukturen und dynamische Speicherverwaltung

Es sollen die Mitglieder eines Vereins verwaltet werden. Dazu dient das Feld **mitglieder**, welches Zeiger auf die Mitglieder enthält. Die maximal mögliche Anzahl der Mitglieder ist auf 100 festgelegt worden. Null-Zeiger in **mitglieder** kennzeichnen noch nicht vorhandene Mitglieder. Gegeben sind die nebenstehenden Deklarationen, Definitionen und Anweisungen.

1. Erstellen Sie die Funktion **neuesMitglied**, dass ein neues Mitglied in das Feld **mitglieder** an der gegebenen Stelle mit Namen und Beitrag einfügt.

Beachten Sie, dass **name** eine lokale Variable ist!

```
#define MAXMITGLIEDER 100
struct Mitglied{
    char* name;
    int beitrags;
};
struct Mitglied* mitglieder[MAXMITGLIEDER];

int verwalte(void){
    char name[80];
    ....
    if( neuesMitglied( mitglieder, 7, name, 105 ) != 0 ){
        //Fehlerbehandlung
    }
    ....
    entferneMitglied( mitglieder, 23 );
    ....
}
```

2. Erstellen Sie die Funktion **entferneMitglied**, die ein Mitglied aus dem Feld **mitglieder** an der gegebenen Stelle entfernt. Der entsprechende Eintrag in **mitglieder** soll durch einen Null-Zeiger ersetzt werden.

Aufgabe 14: Funktionen und Felder

1. Erstellen Sie eine Funktion *getValue*. Diese Funktion soll aus einem zu übergebenen Feld *feld* mit Integer-Zahlen das Element *index* auslesen und an den Aufrufer übergeben. Wenn *index* außerhalb der Feldgrenzen liegt, soll der Aufrufer eine Fehlerinformation erhalten.

2. Demonstrieren Sie in der untenstehenden *main*-Funktion, wie der Aufruf Ihrer Funktion *getValue* erfolgen kann. Es soll der Wert des Feldes *feld* an der Stelle *index* abgefragt und mit Hilfe von *printf* ausgegeben werden.

```
int feld[] = {9,8,7,6,5,4,.....}; //Feld mit vielen Werten
int index = 3;
...
int main(void){
```

```
}
```

Aufgabe 15: Ausdrücke

Gegeben seien:

```
#define OFFS( ptr, a )  *(ptr+a)
char c='9'; int a[] = {1,2,3,4,5,6}; int i=1; int v=13;
char* s="EM2012"; double f=2.2;
```

Geben Sie die Ausgaben der nachfolgenden *printf*-Funktionen an. Die Anweisungen werden in der angegebenen Reihenfolge abgearbeitet. Beachten Sie, dass nicht alle Ausdrücke sinnvoll sind.

<code>printf("%x", (char) (f * 4 + 3.5));</code>	<input type="text"/>
<code>printf("%d", (int) (++i * 1.7));</code>	<input type="text"/>
<code>printf("%d", s[4] - 48);</code>	<input type="text"/>
<code>printf("%u", v & (1<<2));</code>	<input type="text"/>
<code>printf("%x", c - '0' > 8);</code>	<input type="text"/>
<code>printf("%x", (unsigned char)-v);</code>	<input type="text"/>
<code>printf("%d", OFFS(&a[3], -2));</code>	<input type="text"/>

Aufgabe 16: Speicherbelegung

Gegeben sind die globalen Variablen für eine 32-Bit ARM7-CPU :

```
char nummern[][4] = {"9876","123"};
char *pnummern = nummern[1];
unsigned short feld[] = {0x98, 0x76,0x12, 0x34};
```

1. Tragen Sie in die nebenstehende Tabelle die Speicherbelegung ein. Nehmen Sie an, dass der Compiler die Variablen beginnend mit *nummern* der Reihe nach ohne Lücken im Speicher ab Adresse 0x1000 aufsteigend ablegt.

	Speicher (Bytes)
0x1000	<input type="text"/>
0x1001	<input type="text"/>
0x1002	<input type="text"/>
0x1003	<input type="text"/>
0x1004	<input type="text"/>
0x1005	<input type="text"/>
0x1006	<input type="text"/>
0x1007	<input type="text"/>
0x1008	<input type="text"/>
0x1009	<input type="text"/>
0x100A	<input type="text"/>
0x100B	<input type="text"/>
0x100C	<input type="text"/>
0x100D	<input type="text"/>
0x100E	<input type="text"/>
0x100F	<input type="text"/>
0x1010	<input type="text"/>
0x1011	<input type="text"/>
0x1012	<input type="text"/>
0x1013	<input type="text"/>
0x1014	<input type="text"/>
0x1015	<input type="text"/>
0x1016	<input type="text"/>
0x1017	<input type="text"/>

Aufgabe 17: Module

Das Modul *timer* soll eine globale Variable *time* exportieren. Das *main*-Modul soll diese Variable benutzen.

1. Ergänzen Sie die dargestellten Module um alle notwendigen Anweisungen, so dass ein sicherer Export der Variable *time* aus dem Modul *timer* erfolgt.
2. Kennzeichnen Sie die Definition von *time*.
3. Kennzeichnen Sie die Deklaration von *time*.

timer.c

```
void tick( void ){  
    time++;  
}
```

timer.h

main.c

```
int main(void){  
    if(time>42){  
        printf("Hallo\n");  
    }  
    return 0;  
}
```

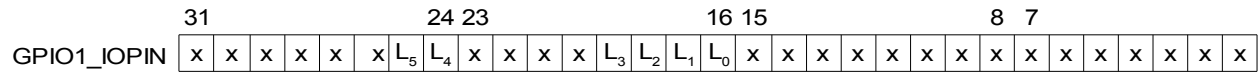
Aufgabe 18: Strings

1. Erstellen Sie ein allgemeines Unterprogramm ***reversestring***, welches die Zeichen eines Quellstrings in umgekehrter Reihenfolge in einen Zielspeicher kopiert. Der Zielspeicher soll vom aufrufenden Programm bereitgestellt werden. Dabei ist es wichtig, dass kein Pufferüberlauf stattfinden kann. Reicht der Platz im Zielspeicher nicht aus, sollen so viele Zeichen wie möglich kopiert werden. Im Zielspeicher soll aber nach Abarbeitung der Funktion auf jeden Fall ein gültiger C-String stehen. Die Funktion soll immer die Länge des gesamten Strings zurückgeben, auch wenn der Speicherplatz dafür nicht ausgereicht hat. Es dürfen keine Bibliotheksfunktionen verwendet werden. Überlegen Sie sich als erstes, welche Parameter die Funktion benötigt.
2. Erstellen Sie die ***main***-Funktion und zeigen beispielhaft die Verwendung der von Ihnen erstellten Funktion. Definieren Sie den Ziel- und Quellstring. Der Quellstring soll den Text „HAW“ enthalten. Es soll überprüft werden, ob die Funktion erfolgreich ausgeführt worden ist.

Aufgabe 19: Bitmanipulation

Es ist eine Funktion zum Ansteuern eines Lauflichts zu erstellen. Das Lauflicht besteht aus den 6 Lampen L_0 bis L_5 . Diese Lampen sind wie unten gezeigt an dem Parallelport GPIO1_IOPIN angeschlossen.

- Ergänzen Sie die Funktion **setzeLauflicht** so, dass die Lampen L_0 bis L_5 entsprechend der Tabelle aktualisiert werden. Schritt ergibt sich aus **schrittnr** modulo 6. Die Ausgabe soll durch direkte Manipulation von GPIO1_IOPIN erfolgen (nicht GPIO1_IOSET oder GPIO1_IOCLR verwenden).



```
void setzeLauflicht(int schrittnr){
```

Schritt	Bitkombination					
	L ₅	L ₄	L ₃	L ₂	L ₁	L ₀
0	0	0	0	1	1	1
1	0	0	1	1	1	0
2	0	1	1	1	0	0
3	1	1	1	0	0	0
4	1	1	0	0	0	1
5	1	0	0	0	1	1
6	0	0	0	1	1	1
usw.						

```
}
```

Aufgabe 20: Strukturen und dynamische Speicherverwaltung

Es soll eine verkettete Liste für die Verwaltung der Fußballnationalspieler erstellt werden.

1. Erstellen Sie die Funktion **addToList**, die einen Spieler in die Liste hinter das Element **spieler** einfügt. Die Funktion soll so aufgebaut sein, dass die Aufrufe, die in **erstellen** gezeigt sind, eine Liste in der Reihenfolge [Startelement, „Gomez“, „Klose“] erzeugen. Beachten Sie dabei, dass die lokalen Variablen nach dem Aufruf von **erstellen** nicht mehr gültig sind!

Ergänzen Sie die for-Anweisung in der Funktion **main** so, dass alle Spieler der Liste ausgegeben werden.

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>

typedef struct Spieler{
    char      *name;
    int       nr;
    struct Spieler *next;
} Spieler;

Spieler *addToList( Spieler* spieler,  char* name, int nr );

//Startelement der Liste
Spieler spielerListe = {.name = "", .nr=0, .next=NULL};

int erstellen(void){
    char name1[] = "Gomez";
    int nr1 =23;
    char name2[] = "Klose";
    int nr2 =11;
    Spieler *spieler;
    spieler = addToList( &spielerListe, name1, nr1 );
    if(spieler==0)return -1;
    spieler = addToList( spieler, name2, nr2 );
    if(spieler==0)return -1;
    return 0
}

Spieler *addToList( Spieler* spieler,  char* name, int nr ){

}

int main( void ){
    Spieler* s;
    erstellen();

    for(
        printf( "
    )
    }
    return 0;
}
```

Aufgabe 21: Makros

1. Erstellen Sie ein Makro INKREMENT(*x*, *max*), das *x* um 1 inkrementiert, solange *x* kleiner als *max* ist.

2. Geben Sie den Code an, der vom **Präprozessor** bei Anwendung der nachfolgenden Anweisungen erzeugt wird:

```
int x=9; int limit=4; INKREMENT( x, limit+2 );
```

3. Diskutieren Sie, ob die nachfolgende Anwendung Ihres Makros erlaubt ist. Falls ja, welches Ergebnis liefern die Anweisungen beim **Ausführen** des Programms?

```
int array[] = {1,2,3,4,5}; int*z=&array[1];  
INKREMENT( z, &array[4] );
```

4. Warum darf Ihr Makro nicht wie folgt aufgerufen werden?

```
int x=9;  
INKREMENT( x-1, 20 )
```

Aufgabe 22: Strings

Erstellen Sie ein Unterprogramm mit folgender Deklaration

```
int input( char **txt );
```

zur Abfrage von C-Strings von der Tastatur. Zur Abfrage der Tastenbetätigungen soll die Funktion **getc** benutzt werden. **getc** gibt als Returnwert den ASCII-Wert der gedrückten Taste oder 0 im Fehlerfalle zurück. Die Funktion **input** soll maximal **COUNT** ASCII-Zeichen im Bereich 0x20 bis 0x7f sammeln und daraus einen gültigen C-String formen. Sie soll beendet werden, wenn Linefeed (0x0a bzw. '\n') gedrückt wurde. Über den Parameter txt soll ein Pointer auf den String zurückgegeben werden. Die Größe des referenzierten Puffers soll **genau** der Stringlänge angepasst sein! Mit dem Rückgabewert sollen die möglicherweise auftretenden Fehlersituationen gekennzeichnet werden.

1. Legen Sie den Rückgabewert für die verschiedenen möglichen Fehlerfälle fest:
0: Kein Fehler
-1:
-2:
2. Es bietet sich an, einen dynamisch angelegten Speicherbereich zu verwenden. Warum darf der zurückzugebende Pointer nicht auf einen in der Funktion **input** definierten lokalen Puffer zeigen?
3. Ergänzen Sie die **main**-Funktion und benutzen Sie dort die **input**-Funktion zur Abfrage, ob das Wort "HAW" eingegeben worden ist. Falls ja, soll **main** mit 0, wenn ein Fehler aufgetreten ist, mit dem Fehlerwert, sonst mit 1 beendet werden. Benutzen Sie zum Vergleichen die Funktion **strcmp**.

```
int main(void){  
    int ergebnis;
```

```
    return ergebnis;  
}
```

Fortsetzung der Aufgabe auf der nächsten Seite!!

4. Ergänzen Sie das nachfolgende **Unterprogramm**:

```
#define COUNT 20  
int input( char **txt ){
```

```
}
```

Aufgabe 23: Ausdrücke

Gegeben seien:

```
#define ADDABS(x,y) ( (y)>0 ? (x+y) : (x-y) )
#define OFFS( a ) a + 7
char c='9'; char[] s = "WM2010"; int i=1; double f=2.2;
int a[]={1,2,-3,4};
int t, v, w, x, y, z;
uint8_t u;
```

Berechnen Sie folgende Ausdrücke unter der Annahme, dass sie in der angegebenen Reihenfolge abgearbeitet werden. Beachten Sie, dass nicht alle Ausdrücke sinnvoll sind.

`t = f * 4 + 3.5;`

`y = ++i * 1.5;`

`v = s[3] - 48;`

`w = ADDABS(3, a[i++]);`

`x = c - '0' > 5 && i++;`

`u = - --i;`

`z = OFFS(3) * 5;`

Aufgabe 24: Zeiger und Speicherbelegung

In einem Programm für eine 32-Bit ARM7-CPU wurden folgende globale Variable definiert:

```
char name[] = "1234";
uint16_t x = 0x1234;
char *pname = &name[2];
```

1. Tragen Sie in die nebenstehende Tabelle die Speicherbelegung ein. Nehmen Sie an, dass der Compiler die Variablen beginnend mit *name* der Reihe nach unter Berücksichtigung eines möglichen Alignments im Speicher ab Adresse 0x1000 aufsteigend ablegt.

Speicher	
0x1000	
0x1001	
0x1002	
0x1003	
0x1004	
0x1005	
0x1006	
0x1007	
0x1008	
0x1009	
0x100A	
0x100B	
0x100C	
0x100D	
0x100E	
0x100F	

Aufgabe 25: Strukturen und dynamische Speicherverwaltung

Es soll eine Bestenliste der deutschen WM-Teilnehmer angelegt werden. Die ersten drei Einträge dieser Liste sind:

NAME	TORE	SPIELE
Mueller	68	62
Klose	50	99
Voeller	47	90

1. Erstellen Sie eine Struktur, in der die Daten eines **einzelnen** Spielers abgelegt werden können. Enthalten sein soll der Name, die Anzahl Tore und die Anzahl Spiele:
2. Legen Sie eine globales Feld an zur Verwaltung der drei besten WM-Teilnehmer der Vergangenheit und initialisieren Sie das Feld mit den oben angegebenen Daten:
3. Erstellen Sie ein Unterprogramm, dass die Gesamtzahl der Tore dieser drei Spieler ermittelt. Das Unterprogramm soll 2 Parameter haben. Der erste Parameter soll die Referenz auf die Spielerliste enthalten, der zweite Parameter soll die Anzahl der Spieler in der Liste angeben.
4. Rufen Sie das Unterprogramm auf (z.B. als Anweisung in der Main-Funktion):

Aufgabe 26: Bitmanipulationen

Aus einem am Mikrocontroller angeschlossenen Gerät wird ein 6-Bit vorzeichenbehafteter Messwert, der im Zweierkomplement kodiert ist, ausgelesen und wie unten dargestellt in einer 8-Bit Variablen temporär als Rohwert gespeichert. Dabei wurden die Bits 6 und 7 zunächst auf 0 gesetzt.

1. Tragen Sie die **dezimale Darstellung** der unten angegebenen Messwerte in die Tabelle ein.
Beachten Sie, dass das Bit 5 das Vorzeichen enthält.
2. Tragen Sie die entsprechende Darstellung als **vorzeichenrichtige 8-Bit Binärzahl** in die Tabelle ein.

Rohwert: 6-Bit Binärwert								Dezimalwert		8-Bit Binärwert							
5				0						7				0			
0	0	0	0	0	0	0	0										
0	0	0	0	0	1	0	0										
0	0	1	1	1	1	1	1										
0	0	1	1	1	1	0	0										

3. Erstellen Sie ein Unterprogramm in C, dass die Konvertierung von 6-Bit nach 8-Bit durchführt.
Dieses Programm bekommt den 6-Bit Messwert in dem Parameter *rohwert* wie in der linken Tabelle dargestellt übergeben und soll den auf 8-Bit gewandelten Wert zurückgeben.

```
int8_t convert( int8_t rohwert ){
```

```
}
```


Aufgabe 27: Programmiersprache C

1. Geben Sie an, wie viele Bytes die nachfolgenden Definitionen in einem 32-Bit System benötigen:

```
char* feld1[]={ "Mueller", "Klose", "Podolski" };  
Anzahl Bytes:
```

```
char feld2[][20]={ "Mueller", "Klose", "Podolski" };  
Anzahl Bytes:
```

2. Wann würde man die zweite Definition in 27.1 bevorzugt verwenden?

3. *feld1* und *feld2* sind wie in 27.1 definiert. Welche Anweisung ist nicht erlaubt? Warum?

```
feld1[1] = feld2[0]; // Anw. 1  
feld2[1] = feld1[2]; // Anw. 2
```

4. Ersetzen Sie die fehlerhafte Anweisung von Aufgabe 27.3 durch eine Anweisung, die die offensichtlich gewollte Aufgabe erfüllt.

5. Innerhalb einer Funktion steht die Anweisung:

```
static int var = 42;
```

Wann wird diese Variable initialisiert?

6. Die globale Integervariable *counter* aus dem Modul *timer* soll auch in anderen Modulen genutzt werden. Geben Sie die Definition und Deklaration an, die in *timer.h* und *timer.c* stehen sollten:

timer.h:

timer.c:

Aufgabe 28: Strings

1. Erstellen Sie ein allgemeines Unterprogramm *strstr*, das prüft, ob in einem String ein anderer Text an einer beliebigen Position enthalten ist. Verwenden Sie keine Funktionen von anderen Bibliotheken. Der Rückgabewert dieser Funktion soll der Index des ersten Zeichens des gefundenen Textes sein. Falls der Text nicht gefunden wurde, soll -1 zurückgegeben werden.
2. Erstellen Sie die *main*-Funktion und wenden dort die von Ihnen erstellte Suchfunktion unter der Annahme an, dass im Puffer *buf* ein gültiger C-String enthalten ist. Suchen Sie in diesem String das Wort „Europameister“.

```
char buf[20000] = " .. hier steht eine Menge Text ... ";
```

Aufgabe 29: Strukturen und dynamische Speicherverwaltung

Es soll eine verkettete Liste für die Verwaltung der Fußballnationalspieler erstellt werden.

1. Erstellen Sie die Funktion *addToList*, die einen Spieler in die Liste hinter das Element *spieler* einfügt. Die Funktion soll so aufgebaut sein, dass die Aufrufe, die in *erstellen* gezeigt sind, eine Liste in der Reihenfolge [Startelement, „Podolski“, „Klose“] erzeugen. Beachten Sie dabei, dass die lokalen Variablen nach dem Aufruf von *erstellen* nicht mehr gültig sind!
2. Ergänzen Sie die for-Anweisung in der Funktion *main* so, dass alle Spieler der Liste ausgegeben werden.

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
```

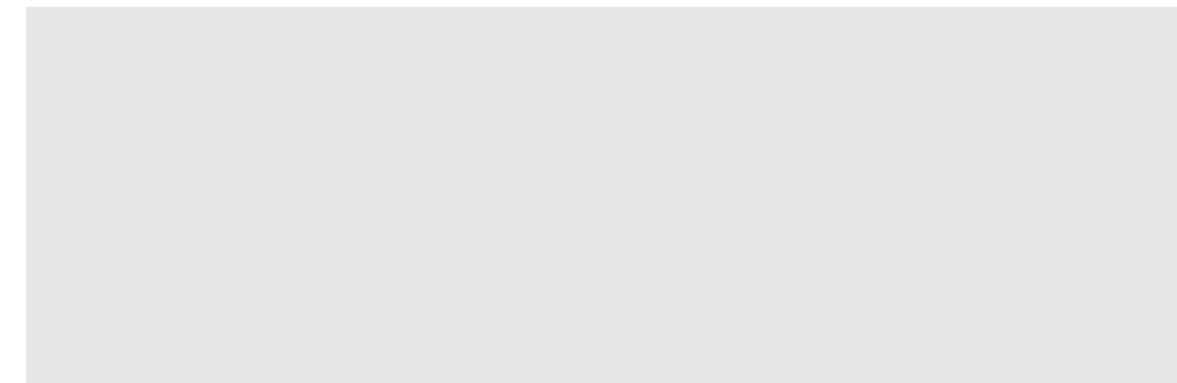
```
typedef struct Spieler{
    char          *name;
    int           nr;
    struct Spieler *next;
} Spieler;
```

```
Spieler *addToList( Spieler* spieler, char* name, int nr );
```

```
Spieler spielerListe = {.name = "", .nr=0, .next=NULL};
```

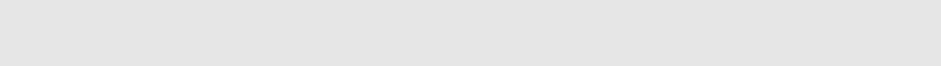
```
void erstellen(void){
    char name1[] = "Podolski";
    int nr1 =20;
    char name2[] = "Klose";
    int nr2 =11;
    Spieler *spieler;
    spieler = addToList( &spielerListe, name1, nr1 );
    spieler = addToList( spieler, name2, nr2 );
}
```

```
Spieler *addToList( Spieler* spieler, char* name, int nr ){
```



```
}
```

```
int main( void ){
    Spieler* s;
    erstellen();
```

```
    for(  ){
```

```
        printf( "%s\n", s->name );
```

```
    }
```

```
    return 0;
```

```
}
```

Aufgabe 30: Strings, Pointer und Speicherbelegung

1. Füllen Sie die nebenstehende Tabelle aus. Kennzeichnen Sie die nicht erlaubten Ausdrücke. Geben Sie für die erlaubten Ausdrücke die Werte an, die sich bei der Programmausführung ergeben würden.

```
char titel1[20]="Master";
char *titel2 ="Bachelor";
char *titel3;
```

	Syntax nicht erlaubt	Wert
<code>sizeof(titel1)</code>		
<code>sizeof(&titel1)</code>		
<code>sizeof(titel1[3])</code>		
<code>sizeof(&titel1[3])</code>		
<code>strlen(titel1)</code>		
<code>strlen(&titel1)</code>		
<code>strlen(titel1[3])</code>		
<code>strlen(&titel1[3])</code>		
<code>sizeof(titel2)</code>		
<code>sizeof(&titel2)</code>		
<code>sizeof(titel2[3])</code>		
<code>sizeof(&titel2[3])</code>		
<code>strlen(titel2)</code>		
<code>strlen(&titel2)</code>		
<code>strlen(titel2[3])</code>		
<code>strlen(&titel2[3])</code>		

2. Nachstehend vier Anweisungen zur Modifikation der Variablen *titel1*, *titel2* und *titel3*. Geben Sie an, welche Anweisungen nicht erlaubt sind. Begründen Sie bitte warum!

```
void unterprogramm(void){
    strcat( titel1, " of Science" ); /* Anweisung 1 */
    strcat( titel2, " of Science" ); /* Anweisung 2 */
    titel2 = "Ingenieur";           /* Anweisung 3 */
    strcpy( titel3, "Diplom" );     /* Anweisung 4 */
}
```

Aufgabe 31: Zeiger und Speicherbelegung

Gegeben sind die globalen Variablen

```
char name[8] = "Otto";
char *pname = &name[4];
char **ppname = &pname;
```

1. Tragen Sie in die nebenstehende Tabelle die Speicherbelegung ein. Nehmen Sie an, dass der Compiler die Variablen beginnend mit name der Reihe nach ohne Lücken im Speicher ab Adresse 0x1000 aufsteigend ablegt.

Speicher

0x1000	
0x1001	
0x1002	
0x1003	
0x1004	
0x1005	
0x1006	
0x1007	
0x1008	
0x1009	
0x100A	
0x100B	
0x100C	
0x100D	
0x100E	
0x100F	
0x1010	
0x1011	
0x1012	
0x1013	
0x1014	
0x1015	
0x1016	

Aufgabe 32: C-Syntax

Das nachfolgende Programm soll das Maximum einer gegebenen Zahlenfolge bilden. Leider haben sich eine Reihe von Fehlern eingeschlichen, die das fehlerfreie Compilieren und eine richtige Ausgabe verhindern. Beseitigen Sie die Fehler, so dass das Programm ohne Fehler und Warnungen übersetzt werden kann und es eine richtige Ausgabe produziert.

```
#include 'stdio.h'

int zahlenfolge[] = { 3, 5, 1, 6, 8, 2, 10, 7, 5 };
#define N (sizeof(zahlenfolge)/sizeof(zahlenfolge[0]));
#define SETMAX(x)      if( x > maximum )    maximum == x

int main(void){
    int maximum = 0;

    int i = 0;

    while i < N ;
    {
        SETMAX( zahlenfolge[i++] );
    }

    printf('Maximum von den %d Zahlen ist: %f\n', N, maximum );
}
```

Aufgabe 33: Bitmanipulationen

Es soll ein Programm zur Ansteuerung von 8 Lampen entwickelt werden. Die Lampen sind an einem 8-Bit Port angeschlossen. Die einzelnen Bits arbeiten wie folgt:

Bit $b_i = 0$: Lampe i leuchtet

Bit $b_i = 1$: Lampe i ausgeschaltet

Der Port kann mit der Anweisung `out8(LAMPENADR, v)` gesetzt werden. Es gibt **keine** Anweisung zum Auslesen des Ports! LAMPENADR ist die Adresse des Ports, v ist der neue Wert.

Schreiben Sie jeweils eine Funktion zum Ausschalten und eine zum Einschalten einer einzelnen Lampe:

`void lampeAn(int i);`

`void lampeAus(int i);`

Beachten Sie, dass bei Aufruf dieser Funktionen die Zustände aller anderen Lampen nicht verändert werden sollen! Mit Start des Programms sind alle Lampen ausgeschaltet.

Aufgabe 34: Makros

1. Erstellen Sie ein allgemeines, universell einsetzbares Makro `MULT(a, b)`, das das Product von `a` und `b` berechnet.

2. Geben Sie den Code an, der vom *Präprozessor* bei Anwendung der nachfolgenden Anweisungen erzeugt wird:

```
#define DEKREMENT(x, limit)  x > limit ? x-- : x
int x; char s[20]; char* p;
...
p = DEKREMENT(p, &p[0]);
```

3. Diskutieren Sie, warum das Makro aus Punkt 2 wohl die vom Programmierer beabsichtigte Aufgabe nicht erfüllt.
4. Verbessern Sie das Makro aus Punkt 2 so, dass es universell einsetzbar wird. Es sollen auch Anweisungen wie z. B. `y=DEKREMENT(x-3, 0)` möglich sein.

Aufgabe 35: Strukturen und Felder

1. In der Funktion *functionX* hat sich ein Programmierfehler eingeschlichen, der zum Totalabsturz des Programms führen kann. Welche Anweisung enthält den Fehler? Warum kann es zum Absturz kommen?
2. Unter der Voraussetzung, dass die fehlerhafte Anweisung auskommentiert wurde, welche dauerhaften Änderungen an den Daten haben sich nach Abarbeitung der *functionX* in *main* ergeben?

```
int feld[] = {1,2,3};
1 struct T {
2     int len;
3     int* data;
4 } s = {4,&feld[1]};
5
6 void functionX( struct T t ){
7     static int f[] = {10,11,12};
8     struct T t1;
9     struct T* t2;
10    t1 = t;
11    t = *t2;
12    *t1.data = f[2];
13    t1.data = &f[1];
14 }
15
16 int main(void){
17     ...
18    functionX( s );
19    ...
20 }
```

Aufgabe 36: Mehrdimensionale Felder

1. Geben Sie die Unterschiede im Ablauf an, die sich bei der Ausführung von Anweisung 1 und Anweisung 2 ergeben. Nennen Sie zwei wesentliche Gründe, warum *functionA* besser ist:

```
void functionA( char spielfeldA[][4] ){
    spielfeldA[1][2] = 0; /* Anweisung 1 */
}
```

```
void functionB( char* spielfeldB[4] ){
    spielfeldB[1][2] = 0; /* Anweisung 2 */
}
```


Aufgabe 37: Ausdrücke

Gegeben seien:

```
#define OFFS( ptr, a )  &(ptr)[a]
char c='9'; char s[] = "WM2011"; int i=1; double f=2.2;
```

Geben Sie die Ausgaben der nachfolgenden *printf*-Funktionen an. Die Anweisungen werden in der angegebenen Reihenfolge abgearbeitet. Beachten Sie, dass nicht alle Ausdrücke sinnvoll sind.

<code>printf("%x", (char) (f * 4 + 3.5));</code>	<input type="text"/>
<code>printf("%d", (int) (++i * 1.6));</code>	<input type="text"/>
<code>printf("%d", s[3] - 48);</code>	<input type="text"/>
<code>printf("%u", -1 & (1<<7));</code>	<input type="text"/>
<code>printf("%x", c - '0' > 9 i++);</code>	<input type="text"/>
<code>printf("%d", - --i);</code>	<input type="text"/>
<code>printf("%c", *OFFS(&s[3], -2));</code>	<input type="text"/>

Aufgabe 38: Speicherbelegung

Gegeben sind die globalen Variablen für eine 32-Bit ARM7-CPU :

```
char nummern[][5] = {"9876","12345"};
char *pnummern = nummern[1];
unsigned short feld[] = {0x9876,0x1234};
```

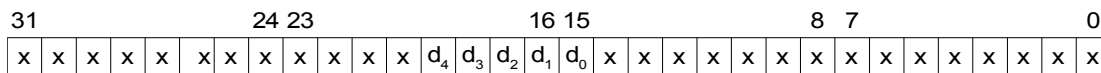
1. Tragen Sie in die nebenstehende Tabelle die Speicherbelegung ein. Nehmen Sie an, dass der Compiler die Variablen beginnend mit *nummern* der Reihe nach ohne Lücken im Speicher ab Adresse 0x1000 aufsteigend ablegt.

Speicher	
0x1000	
0x1001	
0x1002	
0x1003	
0x1004	
0x1005	
0x1006	
0x1007	
0x1008	
0x1009	
0x100A	
0x100B	
0x100C	
0x100D	
0x100E	
0x100F	
0x1010	
0x1011	
0x1012	
0x1013	
0x1014	
0x1015	
0x1016	

Aufgabe 39: Bitmanipulationen

- Ein interner Spannungsmesswert, gespeichert in einer Variable vom Typ *double*, soll an ein externes Gerät als eine 5-Bit Zahl übertragen werden. Die verwendete Kodierung der 5 Bits hat der Hersteller im Datenblatt mit der nebenstehenden Tabelle beschrieben. Die 5 Ausgabebits sind Teil eines 32-Bit Registers, dessen Format unten dargestellt ist.

Spannung	Bitkombination				
1,5 Volt	0	1	1	1	1
...					
0,1 Volt	0	0	0	0	1
0 Volt	0	0	0	0	0
-0,1 Volt	1	1	1	1	1
...					
-1,6 Volt	1	0	0	0	0



Ergänzen Sie die untenstehende Funktion *convert* so, dass entsprechend dem Wert des Parameters *Spannung* die 5 Bits in das Register *ausgabe_port* eingetragen werden. Die anderen Bits sollen dabei nicht verändert werden!

```
volatile uint32_t ausgabe_port;
void convert( double spannung ){
```

```
}
```

- Über eine Datenleitung wurden zwei Bytes empfangen und im Feld *received* gespeichert. Geben Sie die notwendigen Anweisungen an, um daraus eine 32-Bit vorzeichenbehaftete Zahl zu erzeugen. Das Byte in *received*[0] ist das niederwertige Byte.

```
unsigned char received[2];
int32_t value;
```

Aufgabe 40: Modul-Konzept

1. Erstellen Sie ein Modul Spieler, bestehend aus einer .c- und einer .h-Datei. In diesem Modul soll die im Kasten aufgeführte Struktur definiert und eine Funktion **neuerSpieler** implementiert werden. Diese Funktion soll die Elemente der Struktur mit den notwendigen Daten ausfüllen. Die Struktur und die Daten sollen über Parameter der Funktion übergeben werden. Der Speicherplatz für die Struktur soll vom aufrufenden Programm reserviert werden. In dem Kasten unten ist die Verwendung der Funktion gezeigt. Geben Sie alle notwendigen Anweisungen an, damit das Modul zusammen mit der gezeigten main.c fehlerfrei übersetzt und ausgeführt werden kann.

```
struct BasketballSpieler {  
    char* name;  
    char* verein;  
    int   alter;  
};
```

```
/* main.c */  
#include "spieler.h"  
struct BasketballSpieler spieler;  
  
int main(void) {  
    //...  
    neuerSpieler( &spieler,  
                  "Nowitzki", "Dallas", 33 );  
    //....  
    return 0;  
}
```

===== spieler.h =====

===== spieler.c =====

Aufgabe 41: Strings

1. Erstellen Sie ein allgemeines Unterprogramm *appendstring*, welches an einen bestehenden String einen zweiten hinten anhängt. Dabei ist es wichtig, dass kein Überlauf stattfinden kann. Überlegen Sie sich als erstes, welche Parameter die Funktion benötigt. Die Funktion soll auf jeden Fall die Länge des gesamten Strings zurückgeben, auch wenn der Speicherplatz dafür nicht ausreicht. Es dürfen keine Bibliotheksfunktionen verwendet werden.
2. Erstellen Sie die *main*-Funktion und zeigen beispielhaft die Verwendung der von Ihnen erstellten Funktion. Definieren Sie den Ziel- und Quellstring. Der Ziel-String soll bereits den Text „HAW“ und der anzuhängende Quell-String den Text „Informatik“ enthalten.